Chair of Nonferrous Metallurgy

Doctoral Thesis

Clustering and diffusion in aluminium alloys

Dipl.-Ing. Phillip Dumitraschkewitz, BSc

June 2019

Montanuniversität Leoben
Department Metallurgie - Department of Metallurgy
Nichteisenmetallurgie - Nonferrous Metallurgy

Dissertation

Clustering and diffusion in aluminium alloys

by:
Dipl.-Ing. Phillip Dumitraschkewitz

Supervisors:
Assoz.-Prof. Dipl.-Ing. Dr.mont. Stefan Pogatscher
Univ.-Prof. Dipl.-Ing. Dr.mont. Peter J. Uggowitzer

in cooperation with:
Department of Physical Metallurgy and Materials Testing,
Montanuniversität Leoben
Laboratory of Metal Physics and Technology, ETH Zurich
AMAG Rolling GmbH

Leoben, April 2019

AFFIDAVIT

I declare on oath that I wrote this thesis independently, did not use other than the specified sources and aids, and did not otherwise use any unauthorized aids.

I declare that I have read, understood, and complied with the guidelines of the senate of the Montanuniversität Leoben for "Good Scientific Practice".

Furthermore, I declare that the electronic and printed version of the submitted thesis are identical, both, formally and with regard to content.

Date 06.06.2019

Signature Author
Phillip, Dumitraschkewitz
Matriculation Number: 00935425

Acknowledgements

I wish to express my sincere thanks to Univ.-Prof. Dr.mont. Helmut Antrekowitsch, head of the chair, for the possibility to write my doctor thesis at this chair.
I would like to thank my supervisor Assoz.-Prof. Dr. mont. Stefan Pogatscher for his engagement, patience and guidance through this thesis, and the endurance to proof-read manuscipts over and over again. Also I would like to thank Univ.-Prof. Dr. mont. Peter Uggowitzer for his help in writing, input on research direction, hospitality during the stay in Zürich and his endless list of anectodes always good for a laugh.
Dr. Stephan Gerstl is kindly thanked for the hospitality during the stay in Zürich and his support with the cryo-transfer experiments.
The support from the technicians of the chair is very much acknowledged, epecially I like to thank Mr. Erich Troger and Mr. Tom Link for machining the parts for the eletro-polishing setup and their competent advices regarding manufactoring in general. Mr. Bartelme is kindly thanked for his help with sample production and transport.
A big thanks goes out to my colleagues, which make the chair to such a great and friendly working place.

Finally, I would like to thank my family, encouraging and supporting me in all circumstances.

Contents

List of symbols and abbreviations

AA	artificial aging
APFIM	atom probe field ion microscopy
APT	atom probe tomography
AQ	as-quenched
BH	bake-hardening
DB	Doppler broadening
DC	direct current
DSC	differential scanning calorimetry
FIB	focused ion beam
FLANN	fast library for approximate nearest neighbors
FSAK model	Fischer-Svoboda-Appel-Kozeschnik model
FWHM	full-width at half maximum
GP	Guinier-Preston
GPB	Guinier-Preston-Bagaryatsky
HRTEM	high resolution transmission electron microscopy
ICF	image compression factor
kNN	k^{th} nearest neighbor
LEAP	local electrode atom probe
LN_2	liquid nitrogen
NA	natural aging
NMR	nuclear magnetic resonance
PA	pre-aging
PAS	positron annihilation spectroscopy
PALS	positron annihilation lifetime spectroscopy
PLC	Portevin Le-Chatelier
RDF	radial distribution function
RT	room temperature
SANS	small angle neutron scattering
SAXS	small angle x-ray scattering
SDM	saptial distribution map

SEM	scanning electron microscopy
SHT	solution heat treatment
SRS	strain rate sensitivity
SSSS	super-saturated solid solution
TEM	transmission electron microscopy
tof, ToF	time-of-flight
UHV	ultra-high vacuum
WQ	water quenched

Chapter 1

Abstract / Kurzfassung

This thesis gives an overview of existing information on natural aging and clustering in aluminum alloys. An in-depth understanding of existing analysis methods of clustering is established. Moreover, an approach is developed, and applied, to access natural aging times below one hour via atom probe tomography. With the realized experiments it is shown that the as-quenched state can be investigated. Moreover it is demonstrated that clustering and all non-equilibrium vacancy controlled diffusionional processes are size-dependent and natural aging is effectively stopped when the material dimensions reach the nanometer scale.

Diese Arbeit gibt eine Übersicht über existierende Informationen hinsichtlich Kaltauslagerung und Clusterbildung in Aluminium Legierungen. Es wird ein detailliertes Verständnis von existierenden Methoden zur Clusteranalyse erarbeitet. Weiters ist ein Lösungsansatz entwickelt, und angewandt worden, um Kaltauslagerungszeiten kleiner einer Stunde mittels Atomsondenmessungen zugänglich zu machen. Mit den durchgeführten Experimenten konnte aufgezeigt werden, dass auch der abgeschreckte Zustand des Materials untersuchbar ist. Weiters hat sich gezeigt, dass alle Diffusionsprozesse, die auf Nicht-Gleichgewichtskonzentrationen von Leerstellen basieren, größenabhängig sind und die Kaltauslagerung effektiv gestoppt wird, sobald die Materialdimensionen die Nanometerskala erreichen.

Chapter 2

Introduction

> *"In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move."* – Douglas Adams, The Restaurant at the End of the Universe.

Aluminum (Al) alloys are nowadays an important technological material and spread over a broad strength spectrum ranging from high-strength for aerontautic applications, to middle and low strength alloys for automotive, profile or other applications. [1]
The main hardening mechanism for middle- and high-strength Al alloys is precipitation hardening. The applied principle is to solve alloying elements in the α-matrix at a high temperature (solution heat treatment), quench fast enough to form a super-saturated solid solution (SSSS), and further apply a heat treatment at elevated temperature (artificial aging, AA), to form metastable phases in the size of several nanometers (precipitates). Besides that a super-saturated solid solution forms, also a non-equilibrium vacancy fraction is conserved by fast quenching. Only substitutional alloying elements (Si, Cu, Zn, Mg) with also Al are used for classical Al alloys for precipitation formation. Hence the main diffusional process is vacancy movement. If age-hardenable Al alloys are stored at room temperatures an hardness increase is obtained, natural aging (NA). The hardness increase is explained by a precipitation-like process, where small aggregations of solute atoms (below to few nanometers in size) are formed – so-called clusters. The formation of clusters is possible due to the enhanced diffusion by an increased vacancy fraction. [1]
While natural aging occures for all age-hardenable Al alloys, it has an especially large technological impact in AlMgSi alloys. AlMgSi alloys are important for the use as structural material, on automotive or profile applications, due to their cost-effectiveness, good formability and potential hardening ability. A typical production route, for example an AlMgSi automotive sheet, includes the production of the sheet itself via rolling, shipping to the part producer for shaping and the final artificial aging heat treatment by a paint-bake cycle. Natural aging in AlMgSi alloys causes often a negative effect on mechanical properties,

when the material is further artificially aged. Peak hardness is lowered and the artificial aging kinetics are slowed, if the material is stored at room temperature, before the final artificial aging treatment is applied. [1–3]

2.1 Problem & Aim

High resolution techniques such as transmission electron microscopy techniques were not able to identify the origin of the natural aging and its negative effect in AlMgSi alloys, which already occures within about minutes of room temperature storage. Two statements were concluded, the formed clusters do not show ordering, and clusters are very small. Based on the fact that not even with high resolution transmission electron microscopy (HRTEM) clusters in AlMgSi alloys could be investigated, due to the similar atomic masses of Al, Mg, and Si, the further development of the atom probe tomography (APT) technique made it "the" direct investigation tool of choice for clusters. The advantage of APT lies in the easy discrimination of Mg, Si and Al atoms within a three-dimensional reconstruction, which closely corresponds to the real sample geometry. [3, 4]
While a large literature basis on clustering in AlMgSi alloys and the "negative effect" generally already exists (see chapter 3 [3]), the direct observation of the early stages of clusters remains unclear, due to the contradictionary published findings. An obstacle in APT for investigation of early stages of clusters is that from the material a sample needs to be manufactured and transferred to the ultra high vacuum (UHV) system of the atom probe, which both is done at room temperature where natural aging occurs. Therefore earliest investigated natural aging states in the literature correspond to 60 to 100 min.
Aim of this work is to access the time-region below 100 min of natural aging via atom probe tomography, to investigate the early stage of clustering and identify pitfalls causing the contradictionary results in literature. [3]

2.2 Structure of the thesis

An extensive literature survey on existing information of clustering in Al alloys was realized within a review paper given in chapter 3. Which serves as an introduction to the details of the topic. Due to the need of a special sample production strategy and a careful assessment of the gained data, the used methods and sample production process are in detail discussed in chapter 4. Scripts developed and used for analyzing purposes are reported in the appendix, chapter 8. The applied cryo-transfer to the atom probe is described within chapter 5 together with the first gained results on as-quenched AlMgSi alloys. In chapter 6

results of naturally aged APT specimen (*in-situ*) are reported, compared to specimen naturally aged at bulk dimensions, and a possible explanation for contradictionary results for early stages of clustering in literature is presented. Finally, a summary and outlook is given in chapter 7.

Chapter 3

Clustering in age-hardenable aluminum alloys*,**

This review gives an overview of the effects of clusters in various aluminum alloys. Characterization methods are discussed in general and results for the important AlMgSi alloys are presented in detail. Indirect characterization methods, such as hardness, tensile testing, electrical resistivity, differential scanning calorimetry and positron annihilation spectroscopy are discussed, as well as atom probe tomography for the direct measurement of clusters. A particular focus is set on atom probe tomography, where possible artifacts influencing the cluster measurements as well as different cluster finding methods are summed up. A comprehensive summary of investigated alloys and cluster algorithm parameters is given. Moreover, the findings in AlMgSi alloys regarding clusters and changes upon different heat treatments are discussed, starting from early to the latest works. Drawn conclusions are discussed and compared to give a résumé.

*Chapter 3 was already published in [3].

**Thankfully, this research is supported by the Austrian FFG Bridge project, number 853208 (P.D.). S.P. acknowledges financial support by the Austrian Federal Government and the Styrian Provincial Government under the frame of the Austrian COMET Competence Center Programme (K2 Competence Center "Integrated Research in Materials, Processing and Product Engineering", Project A3.31). AMAG Rolling GmbH is kindly thanked for financial support and discussion.

3.1 Introduction

Low density, a broad strength spectrum ranging from 70 to 800 MPa, non-toxicity, high thermal conductivity, high electrical conductivity and a wide range of forming and working processes are just a few reasons why aluminum alloys are currently used [1]. Minimizing energy consumption is the driving force for the application of lightweight materials. Hence one of the main application areas for aluminum alloys is transport. [5] Aluminum alloys typically have a low Hall-Petch coefficient, and therefore their strength cannot be increased extensively for technical applications via grain refinement. In addition, no allotropic phase transitions are known for aluminum at ambient pressure, which restricts hardening strategies in comparison with steels. Solid solution strengthening is indeed used as a strengthening mechanism in low to medium strength alloys. However, aluminum generally shows a tendency towards low miscibility with many other metals in thermal equilibrium. Often intermetallic phases with alloying metals form even at low alloying content. Together with the formation of metastable phases, which can be efficient barriers to dislocation movement, this is the basis for high strength aluminum alloys with age hardening as their dominant strengthening mechanism. Among these alloys the AlCuMg (2000) and AlZnMg(Cu) (7000) alloy series offer the highest strength. They are mainly used in aeronautic applications. The AlMgSi(Cu) (6000) alloy series shows somewhat lower strength, but exhibits a favorable properties profile (including good formability, weldability and corrosion behavior) which makes it the commercially most important of the three classical groups of age hardenable aluminum alloys. All these alloys have been reported to show clustering of super-saturated solute atoms in the very early stages of aging. The phenomenon has huge technological impact and has been intensively addressed in academia and industry over the last decades. However, the term "cluster" is not always well defined in literature and sometimes depends on the characterization method. Note that here we define clusters to be a homogenous decomposition (local aggregation) of alloying atoms, without a detectable structure or ordering [1, 6]. In this review we focus primarily on two alloy classes where, particularly from the technological point of view, clustering plays the most important role: AlMgSi(Cu) and AlCuMg [1, 6–8]. The equilibrium phase common in the AlMgSi(Cu) series is Mg_2Si (β); with additional small amounts of alloyed Cu the Q phase is also prominent [1]. FIG. 3.1 [9] shows several 6000 series alloys and their Si and Mg concentration ranges. A representative Si-rich alloy is AA 6016, with typical 1.2 at.-% Si and 0.5 at-% Mg, in contrast to a typical Mg-rich alloy AA 6061 (0.9 at-% Mg and 0.6 at-% Si). A low content alloy AA 6060 is also shown. We define a balanced alloy as having an Mg/Si ratio of approximate unity and stronger deviating values whether Mg- or Si-rich. The alloy properties are determined by the overall alloying element content, the Mg/Si-ratio, and possible Cu additions which alter the properties even at low content. AA 6016 is deployed in car body sheets, and 6061 can be used in bicycle parts and even in aerospace applications. These two alloys are typical examples where clusters have

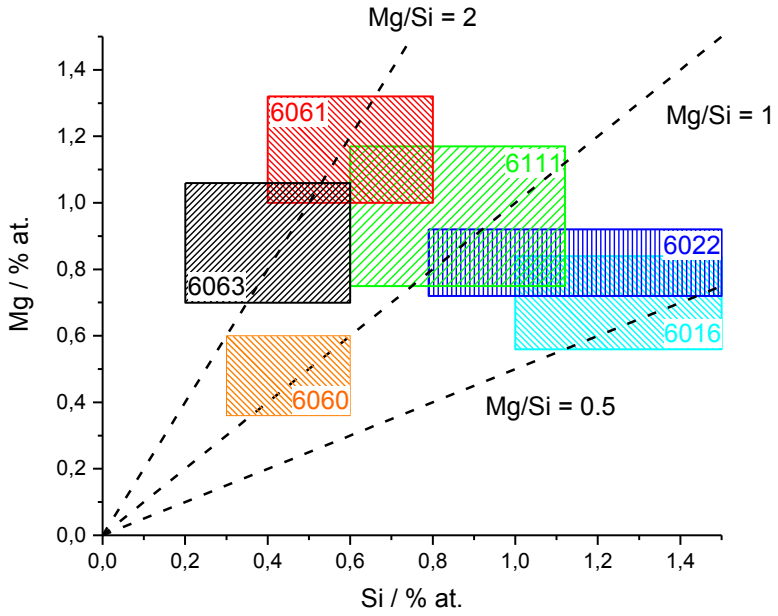

Figure 3.1: Approximate compositions ranges of commercial 6000 series alloys, with guidelines for different Mg/Si-ratios. Redrawn with permission from [9], ©2012 The Japan Institute of Metals and Materials.

a strong negative effect on artificial aging performance: their presence slows down precipitation kinetics and reduces the achievable strength. However, there are also other alloys in the 6000 series (with low Mg and Si content, such as AA 6060) where clusters can have a positive effect on artificial aging. [10–12] For the AlCuMg alloys – depending on the Cu/Mg ratio – the phases Al_2CuMg (S), Al_2Cu (θ) or a mixture of these is formed at equilibrium. Typical alloy compositions of the AlCuMg series where clusters do play an important role are in the range of approximately 1.1 - 1.6 at.-% Cu [1] and Mg $>$ 0.5 at.-% [13]. A typical AlCuMg-based alloy is AA 2024. Interestingly it also is an AlCuMg alloy in which Wilm first discovered age hardening. [8, 14] Applications of AA 2024 are in airframe construction, for example in fuselage/pressure cabin skins or lower wing covers [15]. Generally, the 2000 series of alloys is deployed in aerospace due to a high strength to weight ratio. Clustering in AlCuMg alloys is important, because it is the reason for the "rapid hardening" observed during early artificial aging in these alloys. [8, 16] In the following we address the main aging treatments and the complex role which clusters can play there. The effects of natural aging, pre-aging and artificial aging, and their interdependent influences, are discussed in detail. We also address the techniques which have been used to indirectly and directly characterize clustering during the very early stages of aging in aluminum alloys. We start with indirect characterization techniques such as resistivity measurements, calorimetry, hardness and tensile tests and positron annihilation. Although long in use, these techniques nevertheless

require assumptions and models to link the presences of clusters to studied properties. The only technique which enables direct imaged observations of clusters is often atom probe tomography. This technique, its capability, the issues and the most important results gained are a major part of this review. Aging phenomena Aluminum alloys frequently undergo a homogenization treatment subsequent to casting, followed by a thermomechanical treatment. If we assume, as an example, that our final product is a sheet, thermomechanical processing will include hot rolling and perhaps inter-pass annealing; cold rolling and adjusting of surface roughness; solution heat treatment and quenching; possibly pre-aging; and finally cleaning and surface passivation. Solution heat treatment is used to dissolve the alloying elements in the aluminum matrix (solid solution) and to increase the vacancy concentration. Quenching is applied to freeze in this condition to room temperature in order to form a supersaturated solid solution, which can be used as potential for precipitation hardening. This occurs in various distinct stages of phase transition. Often the earliest stage is clustering, which can have a significant impact on the final material properties. [1]

3.1.1 Natural aging

The as-quenched material has a volatile nature, and exhibits diffusion induced changes even at room temperature (RT). When the material is stored at RT its hardness increases over time. This process has been dubbed "natural aging" (NA), and is caused by clustering of the solute atoms. Clustering at RT occurs in all age-hardenable Al alloys (2000, 6000 and 7000), but is technologically most important in the 6000 alloys. This is discussed in detail in Section 2.4, "Phenomenological description of the effect of clusters". Alloys of types 7000 and 2000 also show NA, where hardness increases during storage at RT. In the 2000 series alloys the natural aged state is in some cases used directly for application. In the 7000 series clustering during NA is of minor importance because it is not an applied materials state and has no strong implications for final properties.

3.1.2 Artificial aging

Aging at elevated temperatures is called "artificial aging" (AA). The influence of clustering on hardness development during AA varies widely among the different alloy families. While in the 6000 series clustering due to prior NA often has a deleterious effect on the AA response, clustering is important for 2000 series alloys because of the occurrence of the rapid hardening effect at AA temperatures. For 7000 series alloys no significant influence of clustering on final AA properties is known.

3.1.3 Conventional precipitation sequences

Typically the phase transitions which occur during aging are simplified by using sequences of different precipitates with increasing thermodynamic stability or decreasing formation kinetics. Recent investigations summarize the precipitation sequence in AlMgSi alloys, as follows in Equation 3.1 [17, 18]:

$$\begin{aligned}\text{SSSS} \rightarrow \text{atomic clusters [7]} \rightarrow \text{GP zones(pre } \beta'') \text{ [19, 20]} \rightarrow \beta'' \text{ [20, 21]} \rightarrow \\ \beta' \text{ [22]}, U1 \text{ [23]}, U2 \text{ [24]}, B' \text{ [25]} \rightarrow \beta, Si(\text{stable})\end{aligned} \tag{3.1}$$

This precipitation sequence should be interpreted as the chronological occurrence of the dominant metastable phases at isothermal aging at different temperatures for a certain amount of time, increasing from low (NA) temperatures for atomic clusters, over β'' at the AA temperature to the equilibrium phases of β and Si at high temperatures or very long ageing times. However, this is also often observed in linear heating experiments (e.g. using differential scanning calorimetry). A recent detailed tabulated overview of the different phases which occur is given in reference [26]. U1, U2, Si (diam.) and B' are present in Si-rich alloys in a typical overaged state [26]. β' is present at peak hardness and overaged states in Mg or Si rich alloys; it forms from β'' or on dislocations [22, 26, 27]. For AlMgSi alloys with Cu, L, "S", C, Q' and Q phases also occur, where L, "S" and C are predecessors of Q,' which is present at overaged and artificially aged states where Q is the overaged (equilibrium) phase. [26] It has been shown that at peak hardness a Cu-containing alloy (0.3 % Cu), β'' and pre- β'' only account for 30% of precipitates; the rest is Cu-containing GP and the precursor of the Q' [28, 29] phase. [30] A suggested precipitation sequence is given below in Equation 3.2 [30]; QP is possibly related to L, "S" is possibly related to QC ("S" should not be confused to the S phase in AlCuMg alloys) [26]:

$$\text{SSSS} \rightarrow \text{atomic clusters} \rightarrow \text{GP zones} \rightarrow \beta'', L, QP, QC \rightarrow \beta', Q' \rightarrow Q \tag{3.2}$$

For AlCuMg alloys the precipitation sequence was constructed, as follows in Equation 3.3 [26, 31]:

$$\text{SSSS} \rightarrow \text{atomic clusters} \rightarrow S'' \rightarrow S' \rightarrow S \tag{3.3}$$

There are different opinions on the exact precipitation sequence and the intermediate phases [8, 26, 32, 33], and also on their impact on the rapid hardening effect. Earlier works declare GPB (Guinier–Preston–Bagaryatsky) zones to be the cause of the rapid hardening effect [31]; later Cu-Mg clusters are explained as the source [8], although in some publications the two are used synonymously [32]. For longer AA times overlapping S (or S′) phase formation is reported [32]; their relative volume fraction is low in the plateau region, but becomes dominant in the peak-aged state. The designations S″ [34] and S′ phase as precursor phases of the S phase is controversial, especially for the S′ phase due to its strong similarity

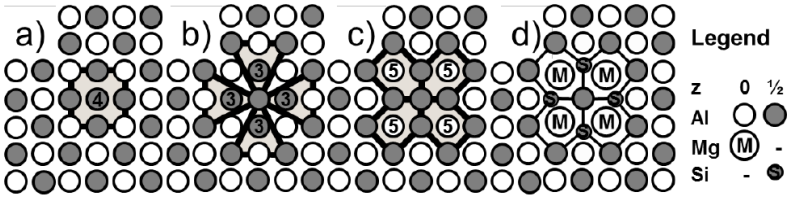

Figure 3.2: Column arrangement principles following from a line defect: (a) If a [001] column segment (4) is moved to interstitial positions (z=0 → 1/2), it is surrounded by (b) four coloumns of same height (3) and (c) four columns of opposite height (5). The atoms marked 3 and 5 have 9 and 15 NNs, respectively. (d) Columns with fewer NNs fit smaller elements (Si or Cu) and columns with more NNs fit larger atoms (such as Mg). Columns with small and large atoms obtain 3-fold (Si/Cu) and 5-fold (Mg) surroundings, respectively. [37] Reproduced with permission from [37], ©2017 Trans Tech Publications.

to the structure of S [26]. For completeness it should further be mentioned that in Mg-rich AlCuMg alloys the T phase [35] can also occur, although it is "rarely found in commercial applications" [26]. For a long time a common general concept was sought regarding the formation of hardness-relevant phases in the different alloying classes, due to the recurrence of similar crystallographic features [36]. An interesting approach to this issue is briefly described as follows. Recent analyses of metastable precipitates in the AlMgSi and AlCuMg systems have shown that the columns along the <100> Al extension follow "the same, simple arrangement principles; columns of large Mg atoms obtain a 5-fold surrounding, while the smaller (Si, Cu) show a 3-fold surrounding." A possible model explaining this behavior is shown in FIG. 3.2 [37]. This principle uses a line defect, by which a segment of <100> Al column is moved half of the conventional unit cell size in the appropriate <100> direction. Therefore, atoms are moved to the octahedral interstitial position. The fcc crystal is now partitioned in columns of 15 and 9 nearest neighbors (NN), instead of 12. The sites with 9 NN would fit smaller atoms like Si or Cu, and the sites with 15 NN would fit larger atoms like Mg. Such a defect needs only one vacancy, and ordering of such defects can explain, for example, the structure of GPB and β'' phase. DFT calculations suggest ordering of Si or Cu, followed by Mg, prior to the defect. If the defect is produced, the direction of the precipitate is fixed. [37] Interestingly, this accords with previous statements which outline the importance of free excess vacancies in the formation of β'' [38] and the latter's interference with natural aging via the available concentration of free excess vacancies determined via the vacancy-prison mechanism [12]. 2.4 Phenomenological description of the effect of clusters While AlCuMg alloys show no essential influence of NA on subsequent AA, AlMgSi alloys exhibit a clear effect. This was already noticed by Brenner and Kostron in 1939 [2]. They observe that for a 0.89 at.-% Mg, 0.77 at.-% Si alloy the yield strength ($\sigma_{0.2}$) increases about 50% within three days (FIG. 3.3 [2]). After NA for 7 days they obtain a much slower hard-

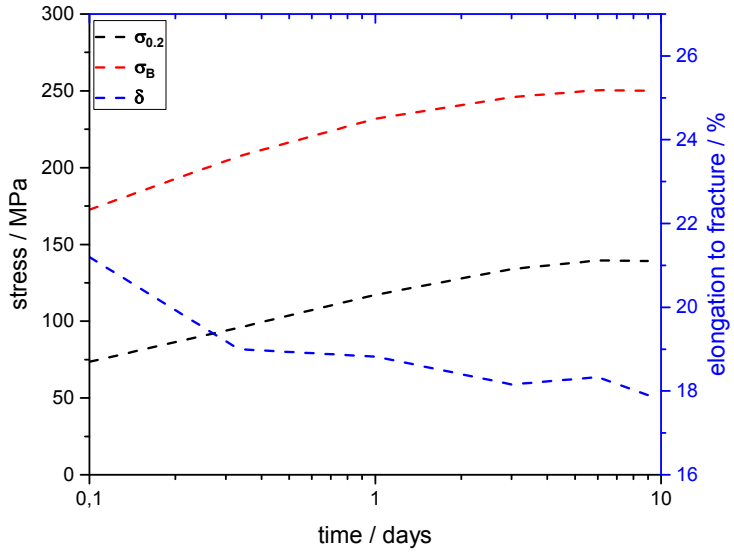

Figure 3.3: Evolution of mechanical properties for increasing natural aging time for a 0.89 at.-% Mg and 0.77 at.-% Si alloy. $\sigma_{0.2}$ is the yield strength and σ_B the ultimate tensile strength and δ is the elongation to fracture. Redrawn with permission from [2], ©2018 Carl Hanser Verlag GmbH & Co.KG München.

ening reaction and a lower maximum of yield and ultimate tensile strength upon subsequent AA compared to direct AA without NA [2] (FIG. 3.4). Natural aging is also seen to increase the activation energies of subsequent AA precipitation kinetics [12]. This undesirable effect on the mechanical properties of NA is later called the "negative effect" in literature, and has its origin in the clustering which occurs during NA [4]. Note that some lean AlMgSi alloys with low strength can also show a positive hardening effect of RT storage on subsequent AA [11, 39]. Brenner and Kostron investigated the effect of a pre-aging (PA) treatment directly after quenching. They observed that PA slows down hardening at RT and accelerates the aging response at elevated temperatures (FIG. 3.5). It is proven that with PA the negative effect can be significantly reduced; nowadays this is the basis for industrial pre-aging treatments in the production of 6000 series alloys used for automotive body panels [10, 40]. Brenner and Kostron also stated that even small amounts of Cu may lessen the negative effect, without changing the NA hardness evolution. [2] It was further demonstrated that interrupted quenching to AA temperatures for short periods of up to 10 min can also stabilize the material (similar to PA [41, 42]) and generate a beneficial aging response compared to a quenched and RT-stored material [38]. An interesting approach was applied in [43] with a long-term PA (100°C / ∼25 days) of a Si 0.72 at.-%, Mg 0.78 at.-%, low Cu alloy. The material exhibits a yield strength similar to the T6 condition, but higher tensile strength and significantly higher total and higher uniform elongation [43]. A different strategy to

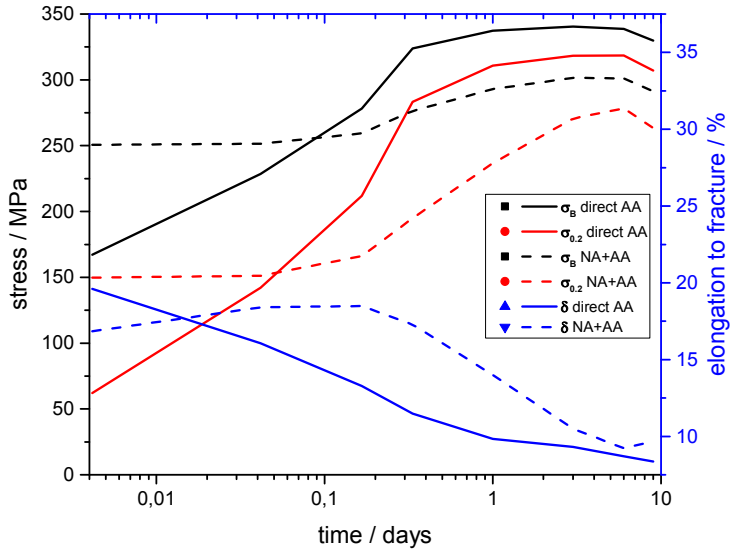

Figure 3.4: Evolution of mechanical properties for direct artificial aging (AA) and AA with prior natural aging of 7 days (NA+AA) for a 0.89 at.-% Mg and 0.77 at.-% Si alloy. $\sigma_{0.2}$ is the yield strength, σ_B the ultimate tensile strength and δ is the elongation to fracture. Redrawn with permission from [2], ©2018 Carl Hanser Verlag GmbH & Co.KG München.

hinder the negative effect of NA, microalloying for AlMgSi alloys with Sn or In, was recently introduced by Pogatscher et. al. [44–46]. The study in [44] demonstrates a significant delay of NA for AA 6061 due to microalloying of Sn (see FIG. 3.6 [44]). The Sn-enhanced material with NA and subsequent AA also reaches the T6 hardness of the non-Sn-enhanced reference, which was artificially aged directly; see FIG. 3.7. For an AA 6061 with traces of Sn an unusually high and fast hardness increase for high AA temperatures (250°C) was obtained, although not reaching full T6 hardness at this temperature. The effect of Sn alloying was phenomenologically similar to the known effect of PA on high AA temperatures [47]. The temperature influence (5 to 45 °C) for NA was also studied for AA 6061 with and without Sn, Sn + In additions. Higher temperatures lead to faster increase in hardness; Sn+In delayed the hardness increase the most. Sn and Sn+In were seen to increase the effective activation energy [45]. When Si is substituted by Ge the NA kinetics is notably retarded [48]. This is also attributed to vacancy solute interaction energies and could be interpreted as an effect similar to Sn. A design strategy for microalloying in AlMgSi alloys was also proposed by Werinos et. al. [49]. They outline the importance of solution heat treatment temperature, to ensure the dissolving of the micro-alloyed elements. A strong detrimental effect of Si on the delaying nature of microalloying NA was observed. A smaller influence of Mg was seen; less Mg led to a delayed increase in hardness. Cu marginally retained NA with higher content. A designed alloy was shown to exceed 180 days RT stable hardness values with

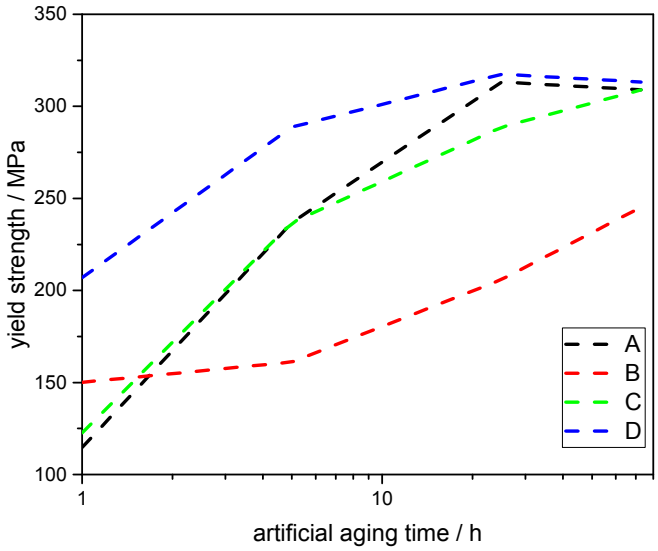

Figure 3.5: Yield strength of an AlMgSi alloy for different heat treatments. Artificial aging and pre-aging at 150°C, given is the total aging time. A) Direct artificial aging. B) 7 days natural aging, subsequent artificial aging. C) 1 h pre-aged, 7 days natural aging and artificial aging. D) 1 h pre-aged, 7 days natural aging, 4 % straining and subsequent artificial aging. Redrawn with permission from [2], ©2018 Carl Hanser Verlag GmbH & Co.KG München.

increased AA response at higher AA temperatures. [49] Various microalloying elements for Al-Cu were investigated in [50, 51]. In this alloying system, too, adding trace amounts of Sn, In and Cd reduces clustering upon NA due to their large binding energy to vacancies and the limited orbit motion of vacancies around these solutes [44]. However, compared with 6000 series alloys the influence of NA on AA is different for 2000 series alloys, since small precipitates are formed serving as nuclei during AA. Moreover, no negative interdependence of NA and AA has so far been reported. Nevertheless, cluster formation is present during NA in 2000 series alloys, which is a material state of application, and AA incorporates an important hardening phenomenon which is linked to clustering [52]. The aging process at elevated temperatures has a two-step nature in 2000 series alloys. FIG. 3.8 shows a typical AA response. The first stage of hardening occurs very rapidly (within about 60 seconds), and generates about 60% of the absolute hardness increase [8]. This technologically important hardening phenomena has been linked to very fast cluster formation and is called "rapid hardening" [33]. Interestingly, this behavior actually needs a minimum of about 0.5 at.-% Mg to be present (FIG. 3.8) [53]. The hardness increases with further increasing alloying content of Mg for the rapid hardening response and the later aging peak. Cu-Cu or Mg-Mg clusters were not found to correlate to the rapid hardening effect, but Cu-Mg clusters. An

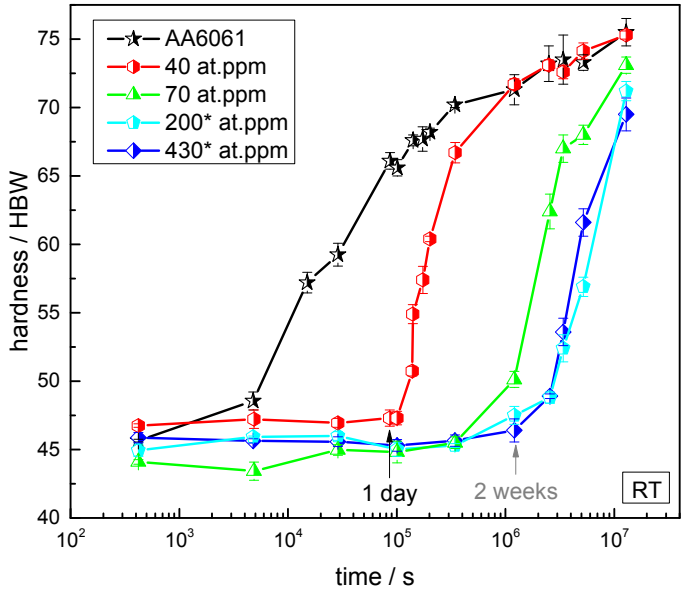

Figure 3.6: Evolution of hardness during RT storage after quenching for the AA 6061 alloy, with and without Sn addition. The increase in hardness is retarded with increasing amount of Sn. Sn additions above the solubility limit (approx. 100 at. ppm) are marked with an asterisk. Adapted with permission from [44], ©2018 by the American Physical Society.

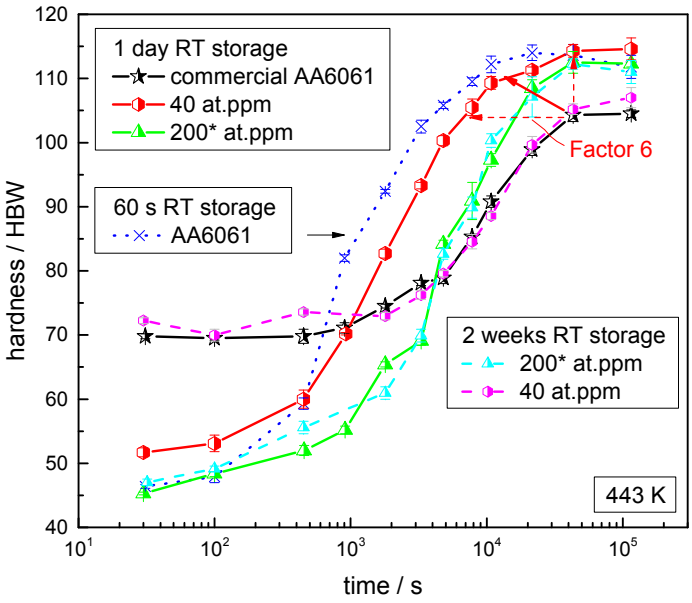

Figure 3.7: Evolution of hardness during artificial aging at 170°C with and without Sn for different prior natural aging times. Minute additions of Sn hinder the negative effect. Sn additions above the solubility limit (approx. 100 at. ppm) are marked with an asterisk. Adapted with permission from [44], ©2018 by the American Physical Society.

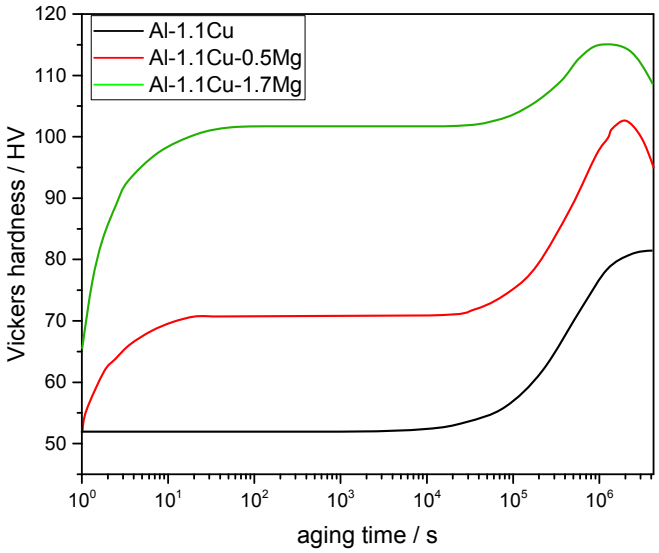

Figure 3.8: Artificial aging at 150 °C. Three different alloys are studied Al-1.1Cu-xMg for x=0, 0.5 and 1.7. Above a critical Mg concentration a rapid hardening effect is seen. Redrawn with permission from [13], ©2018 Elsevier.

approximate ratio of Mg/Cu of 2 and small cluster sizes with high number density are seen as most potent strengthening agents [13].

3.2 Indirect characterization of clusters

The presence of clusters is deduced from various different measurement methods. Often indirect methods are utilized to follow the temporal property changes. "Indirect" means that only the effect of clusters is measured, and not the distribution or size of clusters themselves. In this section we discuss important results regarding electrical resistivity, differential scanning calorimetry, hardness and tensile test findings, and finally positron annihilation studies. Characterization methods are discussed in general and major results for the important AlMgSi alloys are presented in detail. Although we try to sketch the overall picture in the literature, we do not claim completeness.

3.2.1 Resistivity

A very precise way to measure early stages of decomposition in metals is via specific electrical resistivity measurements. Due to the high conductivity of metals, high precision measurement setups are required. An advantage is that in-situ data can be obtained quite easily by measuring over time at a fixed temperature or heating rate. The resistivity signal is sensitive

to structural evolution such as vacancy annihilation, solute depletion, cluster formation, ordering of phases, and precipitation. [54–56] The classical view is that resistivity is dependent on the mean free electron path. This mean free path is disturbed by several sources, i.e. crystal defects (vacancies, grain boundaries, dislocations), solute atoms in the matrix, phase boundaries, ordering in precipitates/phases, and temperature, due to electron-phonon interactions. To reveal changes due to clustering and precipitation effects the resistivity change is usually measured at a fixed temperature. As to precipitation, one would expect decreasing resistivity changes due to decreased matrix solute content, which typically increases the electron mean free path in the matrix; this is indeed obtained for precipitation at AA temperatures. Although counter-intuitive, clusters generate an anomalous resistivity maximum [57] due to increased electron scattering [58], with a maximum cluster size at the scale of approximately 1 nm [56]. In-situ resistivity measurements at several temperatures were performed on Mg-rich AlMgSi alloys decades ago. Large differences in the evolution of resistivity change over time were revealed for different temperatures, as shown in FIG. 3.9 [59], and confirmed by other authors [57]. Most significant is the resistivity increase over aging time in the temperature range (10 – 50 °C) the material behavior changes for temperatures > 50°C [59]. An early fast increase of resistivity is obtained followed by a $\log(t)$ resistivity change. Based on considerations of Hirosawa et al. [60], Zurob et al. [61] concluded that the logarithmic time dependence of the resistivity change is due to cluster growth, which is dominated by vacancy escape. But in later works [62] cluster growth could not be obtained for natural aging, only an increase in number density of clusters was observed with APT (FIG. 3.10). Seyedrezai et al. [63] reports about changes in the slopes of the resistivity over log(t). Several "stages" were obtained, with temperature dependent changes between stages. The relationship of resistivity changes to alterations in mechanical behavior is complex and is alloy dependent as can be seen for hardness change over resistivity change (FIG. 3.11). For the low Si containing alloy in the work of Kim et. al. [9] an increase in electrical resistivity is connected to an increase in hardness, but with high Si content a region ("region 2") is built up where relative large changes in resistivity do not result in increased hardness. Generally higher Si of the alloy leads to lower measured hardness increase for the same electrical resistivity change. A linear relationship was found between the number density of aggregates and the maximum of the electrical resistivity anomaly for pre-aging and artificial aging temperatures. It is concluded that the larger spacing in between clusters results in smaller resistivity anomaly. [64] Generally, the drawback of the resistivity method is the weak connection between signal, cluster development and mechanical properties. On the opposite, the in-situ measurements can be conducted with a high sensitivity and time-resolution, and it has the potential to investigate early vacancy related processes.

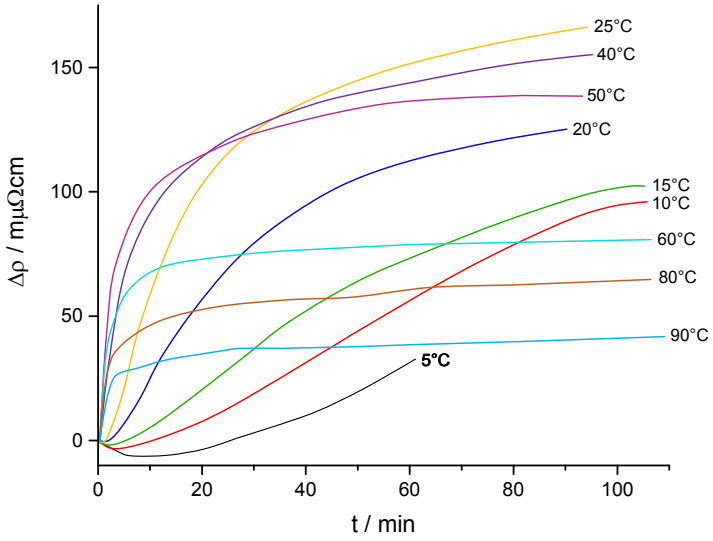

Figure 3.9: Resistivity change over natural aging time for different temperatures. Material (0.60 at.-% Mg, 0.30 at.-% Si and 0.02 at.-% Cu) is quenched to -78°C. Redrawn with permission from [59], ©2018 Elsevier.

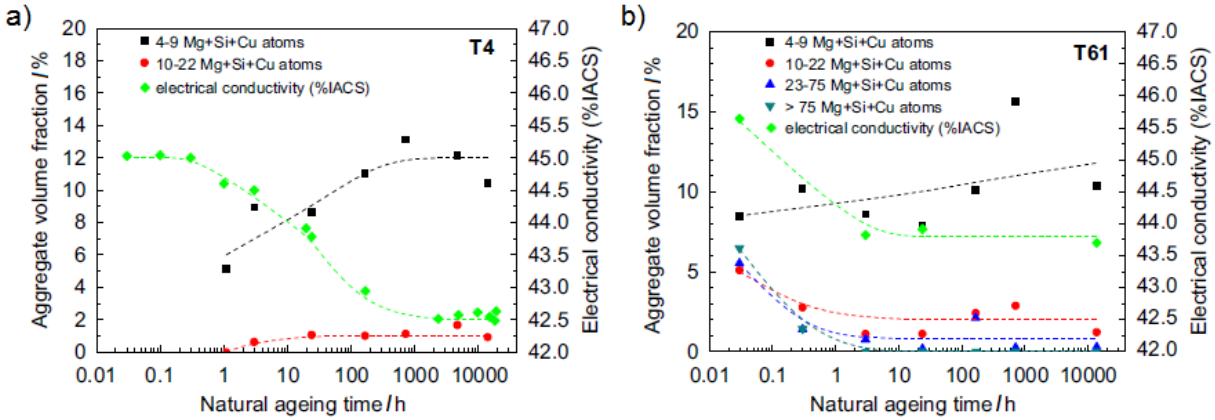

Figure 3.10: Effect of natural aging time on the volume fraction of solute aggregates (Mg, Si and Cu) and on the electrical conductivity in a) T4 and b) T61 condition. Aggregate sizes are binned into 4-9, 10-22, 23-75 and > 75 solutes (not corrected for detection efficiency). Adapted with permission from [62], ©2018 Elsevier.

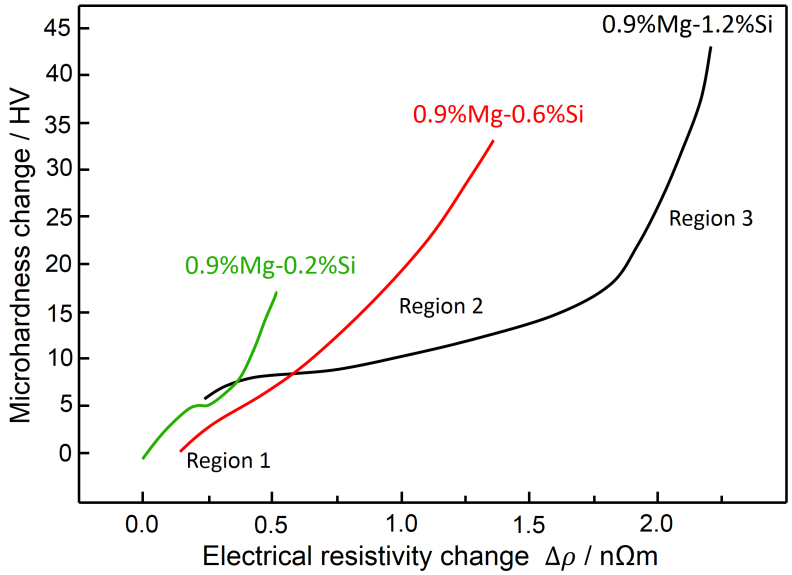

Figure 3.11: Micro-hardness over electrical resistivity change due to Cluster (1) formation at RT for different Si concentrations. Changes are grouped into three regions. Redrawn with permission from [9], ©2012 The Japan Institute of Metals and Materials.

3.2.2 Differential scanning calorimetry

Differential scanning calorimetry (DSC) has been widely used to study phase transitions in metallic systems and has been also used to measure clustering in aluminum alloys [9]. It enables access to thermodynamics and kinetics of phase changes and reveals the heat related to a phase transition in dependence on temperature and/or time. [56, 65] Note that compared to steels, Al alloys usually lack in measureable thermal expansion changes induced by phase transitions, which is why DSC is used much more than dilatometry [66, 67]. However, recently it became possible to observe precipitation reactions in AlMgSi alloys via dilatometry. This was achieved by high stability laser dilatometry measurements, which may also become an interesting method for measuring clustering stages [68]. Generally, DSC experiments measure the difference in the heat flow required to heat a sample mass. The exact measuring procedure depends on the type of DSC used. The sample is measured in a crucible in reference to an empty crucible or a crucible with a reference mass. In case of the very small heat-release associated with clustering in aluminum alloys, the excess quantities are usually measured, i.e. the thermograms of the samples are measured against an equi-mass reference of pure Al. For a detailed description of execution and analysis we refer readers to [69]. In the following, typical results of DSC measurements of AlMgSi alloys are discussed. The Mg-rich alloy AA 6061 was first investigated by DSC by Dutta and Allen [70]. The thermograms show typical cluster formation peaks at ∼50-100 °C

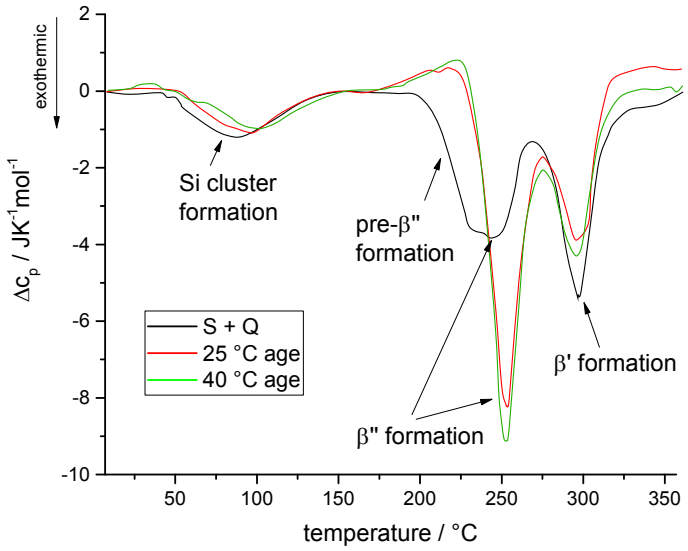

Figure 3.12: DSC thermograms, excess specific heat capacity, of an as-quenched (S+Q), RT (25 °C) and elevated temperature (40 °C) aged AA 6061 alloy (Mg-rich). Redrawn with permission from [70], ©2018 Springer Nature.

and endothermic traces at ∼210 °C due to cluster dissolution. Further clusters formed during NA generate a change in the following precipitation; a right shift of the β'' peak to higher temperatures due to NA is observed. In the as-quenched condition the β'' peak is an overlapping double peak (see FIG. 3.12) and no endothermic traces are found. The exact shape of the thermogram is dependent on the heating rate used, suggesting that the processes are kinetically controlled [71]. Note that the DSC curves are different for Mg-rich and Si-rich alloys; in Si-rich alloys Si-precipitates are formed at higher temperatures and the Mg_2Si phase precipitation is suppressed [72]. The thermogram of an Si-rich alloy for different heating rates is shown in FIG. 3.13 [71]; compare with FIG. 3.12. Several different heat treatments or thermo-mechanical treatments (such as NA, PA, pre-straining and AA or combinations of these) change the appearance of the DSC signal. Important findings which focus on the formation of clusters and their influence on the precipitation sequence are summarized in the following. Natural aging [73, 74]: Usually a double cluster peak is present for the solution heat treated material, and for NA only one of the two peaks is apparent. The double peak nature is attributed to the existence of two kinds of clusters, Cluster 1 (C1) and Cluster 2 (C2) [73]. The exothermic cluster peak was even fitted into three overlaying peaks in [75]. The peak temperatures (C1, C2) and peak areas also depend on the Si and Mg content and ratios [9]. Cluster formation generally increases with increasing Mg and Si alloying content and is most pronounced for the Mg/Si ratio of approximately 1.0 (FIG. 3.14). The C1 reaction was seen to be completed within 60 to 100 min of NA;

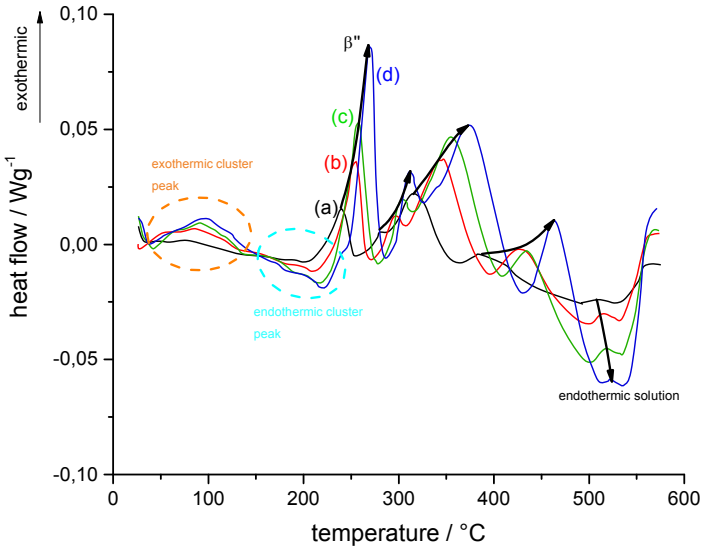

Figure 3.13: DSC thermograms, excess heat flow, of as-quenched Al-0.4%Mg-1.3%Si alloy (Si-rich) for different heating rates. (a) 5°C/min; (b) 10°C/min; (c) 15°C/min and (d) 20°C/min. Additionally guidelines for the peak shifting is added. Redrawn with permission from [71], ©2018 Elsevier.

C2 exists up to one week of NA but is later low in signal [76]. Two important changes with increasing NA time are relevant: a shift of the exothermic β'' peak to higher temperatures and an increase in the β''peak. After roughly one week the DSC traces stabilize [77]. Also in the Cu containing alloy AA 6111, naturally aging shifts the β'' peak to higher temperatures and the exothermic cluster peaks disappear. The Q' peak was found not to be influenced significantly by NA. Although the exothermic cluster peak seems to vanish, the endothermic cluster dissolution peak is present [74]. Pre-aging [10, 73]: The β'' peak is shifted to lower temperatures with increasing PA time at 60°C, as well as decreasing endothermic traces of cluster dissolution, which indicates their increasing stability against NA with PA (FIG. 3.15). For short PA times the exothermic cluster peak (C1, C2) vanishes. With increasing NA after PA the endothermic traces re-appear. [77] Pre-straining [76] results in a left shift of the β'' peak and in a disappearance of endothermic cluster reactions. When the temperature regime of the exothermic cluster reaction is rapidly overcome at a high heating rate the β'' peak activation energy, measured via a Kissinger-like method [65, 78], is significantly lowered. The lower activation energy for the β'' peak demonstrates the essential changes for the following precipitation sequence due to low temperature cluster formation, and shows another DSC experimental verification of the delaying nature of the clustering on further precipitation reactions [79]. The double peak C1, C2 attribution to NA and PA clusters is possibly not as straight forward, since both, NA and PA, consume the exothermic peaks if applied, but

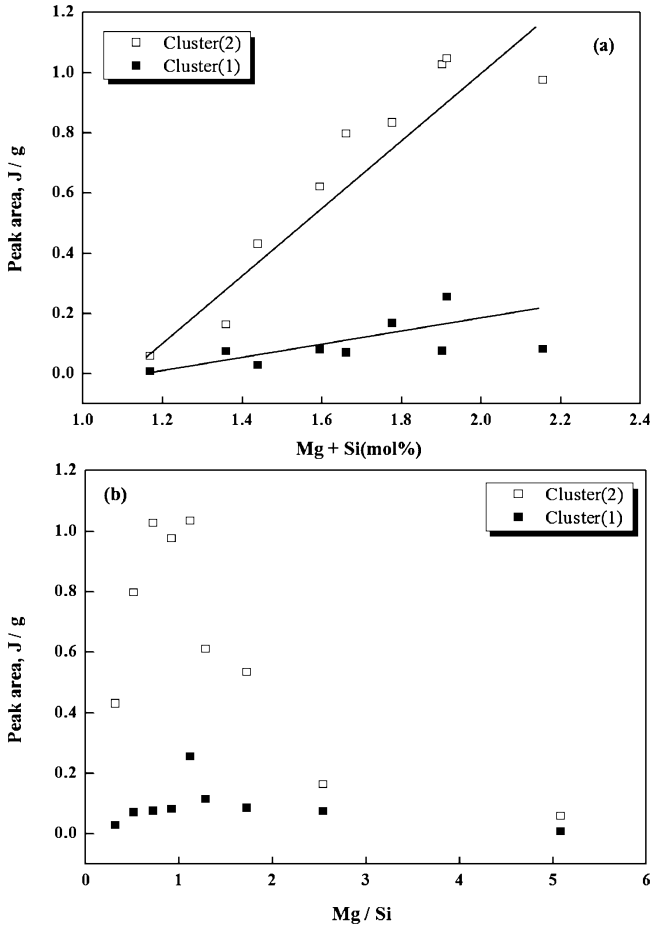

Figure 3.14: Peak areas of Cluster (1) and Cluster (2) as a function Mg+Si at. % and (b) Mg/Si. Reprinted with permission from [9], ©2012 The Japan Institute of Metals and Materials.

increasing endothermic cluster peaks with NA after PA hints in the direction that at NA clusters are formed at RT which are resolved at AA temperature. A direct connection from DSC to mechanical properties can be drawn for the occurrence for the endothermic cluster peak of NA material, where at temperatures around 225 – 250 °C a decrease of hardness is obtained i.e. reversion of NA clusters can be obtained (see section 3.3). Also the delay for short artificial aging times for naturally aged material can be related by the shift of the β'' peak, which for NA material moves to higher temperatures away from usual bake hardening temperatures of approximately 180 °C.

3.2.3 Hardness evolution and tensile test findings

Phase transitions often result in changes in mechanical properties, which is also the reason for using such transitions to optimize engineering materials. An old, but still frequently used method is to follow the change of hardness over time or temperature for a certain heat treatment. Unfortunately interpretation is relatively difficult because the processes moni-

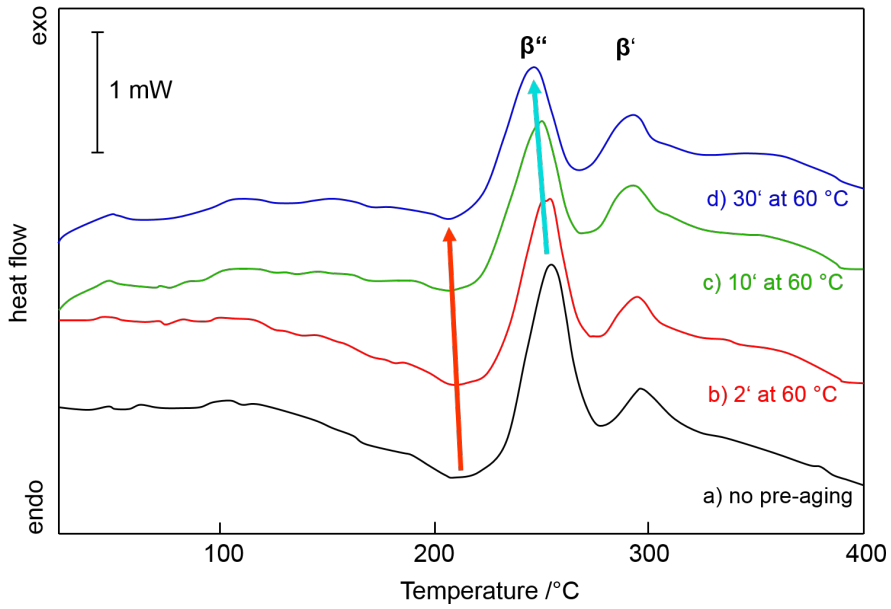

Figure 3.15: DSC thermograms, excess heat flow, of an AA 6016 alloy (Si-rich) for different pre-aging times at 60°C. (a) naturally aged, (b) 2 min, (c) 10 min and (d) 30 min of pre-aging. Redrawn with permission from [10], ©2018 Elsevier.

tored can be complex. Adequate models for hardness are also unavailable, and direct linking of hardness to atomic processes is not possible. Nevertheless, a set of clever experiments in composition-time-temperature space can still help us to understand the underlying processes. [56] Tensile testing is also an established way to measure the decomposition of solid solutions [80]. However, it requires more time and effort than hardness measurements. Note that linear conversion of hardness to yield strength is often performed in literature studies (e.g. [81]). In certain cases this can be done, but experimental justification is almost always needed. [56] For clustering one needs to be aware that models have been discussed for yield strength, but these are still subject to debate [82–84]. The complex kinetic situation of NA followed by AA in 6000 series alloys is demonstrated by hardness measurements in FIG. 3.16 [26], for an almost balanced alloy (Mg/Si $\sim$ 0.87). The AA response for longer holding times (120 – 480 min) at elevated temperatures exhibits a minimum in hardness in the range of prior storage of 30 to 6000 min at RT. The AA response shows a re-increase for longer prior NA times, although the negative effect is not fully restored. Especially the detrimental effect of natural aging for shorter artificial aging times is apparent and therefore most important for bake-hardening treatments. Sometimes the hardness evolution during RT storage is also separated into different stages: see FIG. 3.17. Hardness evolution is often seen to be proportional to $\log(t)$ over RT in the stages, beginning with a stage with practical no hardness increase, followed by an accelerated stage and again a deceleration. Activation energies can be calculated from transitions between different stages. [45] For naturally

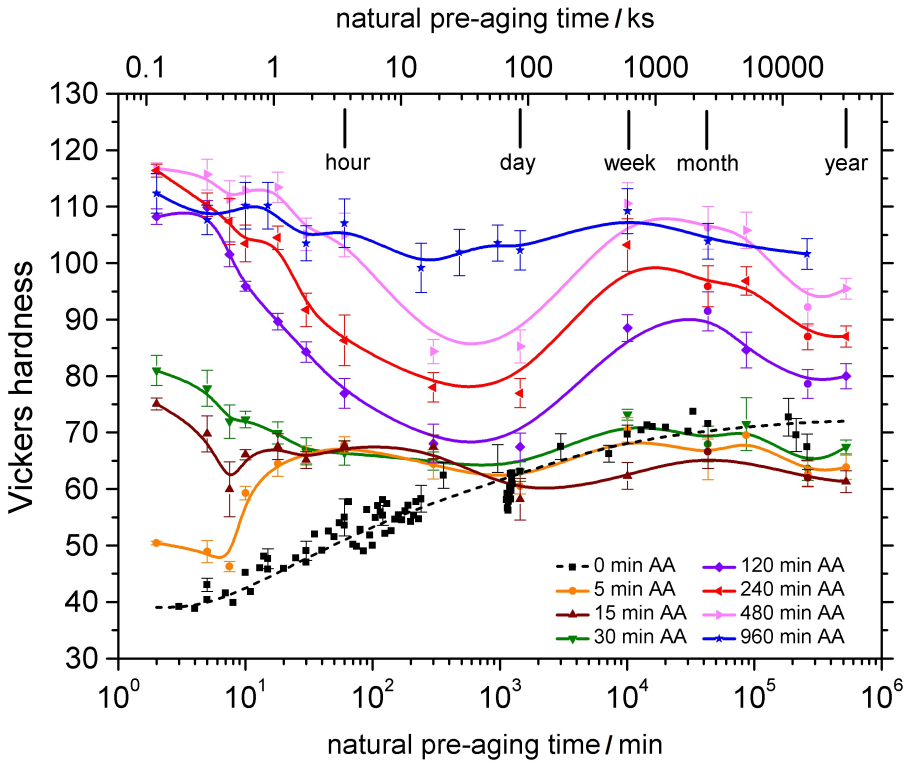

Figure 3.16: Artificial aging response as a function of prior natural aging time in an Al-0.67%Mg-0.77%Si alloy. Natural aging is presented as 0 min AA. [26, 77] Adapted with permission from [26], ©2018 ASM International.

aged material a decrease in hardness for short AA tempering times can often be obtained, as shown in FIG. 3.18 [12]. The AA response for an unusual AA temperature of 250°C is shown. The decrease is interpreted from solving NA co-clusters (some authors refer to this as retrogression or reversion). The cluster dissolution kinetics was studied (AA 6061) via hardness curves, based on a simple model from [74], which states:

$$\Delta H \sim \sqrt{f}, \tag{3.4}$$

where ΔH is the hardness change and f is the relative volume fraction of clusters. This generates a $Q_{diss} = 0.79$ eV [12], in general agreement with values measured via isothermal calorimetry for AA 6111 [74]. However, note that the model used has not been justified via experimental insight in the microstructure [12]. Further, short time reversion treatments at 225 °C for several minutes lowered the yield strength and were shown to almost restore the BH response [85]. In addition to simply following the evolution of strength or hardness upon the formation or dissolution of clusters, an interesting indirect effect of clusters can be seen for tensile test curves. As-quenched (or short NA) AlMgSi alloys exhibit the PLC (Portevin Le-Chatelier) effect [86]. The PLC effect is eliminated with longer NA or PA times, as can be seen for NA in FIG. 3.19 [86] for an excess Si alloy. It is also found that the strain rate

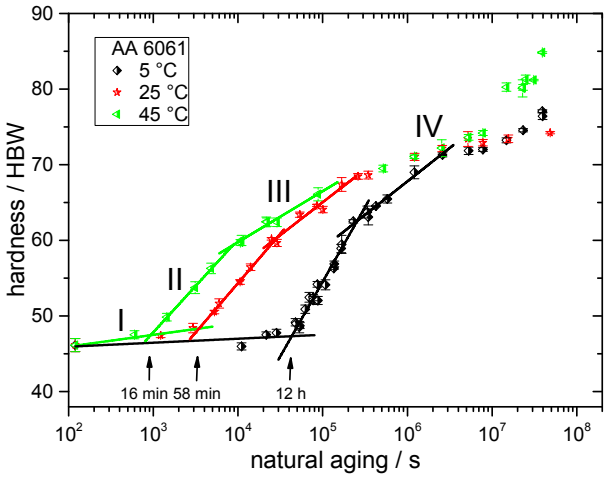

Figure 3.17: Hardness over natural aging for different low temperatures (5, 25 and 45°C) for the alloy AA 6061. Transition in between stage I and II labeled with 16 min, 58 min and 12 h respectively. Adapted with permission from [45], ©2018 Elsevier.

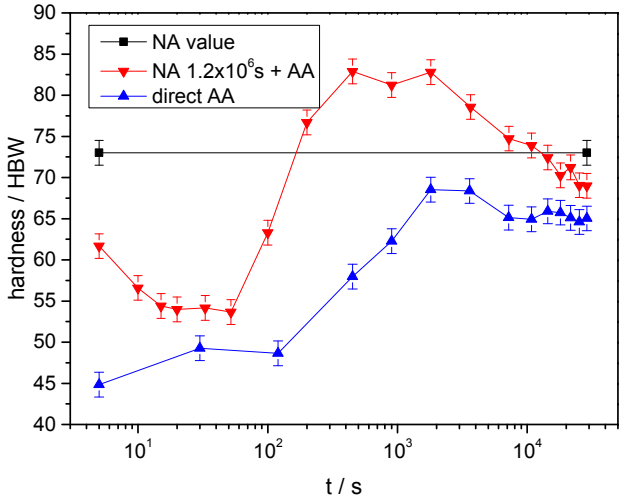

Figure 3.18: Hardness over artificial aging at 250 °C for the alloy AA 6061 without prior natural aging (NA) and with 1.2 × 106 s of NA. Adapted with permission from [12], ©2018 Elsevier.

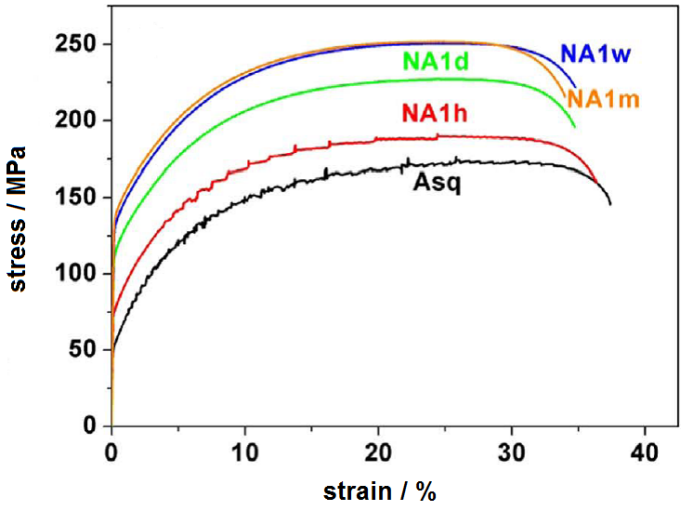

Figure 3.19: Stress-strain curves for Al-1.18%Mg-0.48%Si alloy with increasing natural aging time: Asq – as-quenched, NA1h – one hour natural aging, NA1d – natural aging one day and NA1w – natural aging for one week. Adapted with permission from [86], ©2018 Elsevier.

sensitivity (SRS) is higher for NA samples than for PA + NA samples. The PLC effect was more readily observed for the excess Mg alloy and cannot be eliminated with too short NA or PA in some cases. [86] The effects of clusters on the mechanical properties can be followed by tensile tests and hardness measurements, although no direct conclusions to the cluster form or chemistry can be made. Also no general accepted model for the strength / cluster correlation exists, but kinetics are often interpreted by means of a monotonous function, i.e. increased strength corresponds to more clusters (e.g. the mentioned $\Delta H \sim \sqrt{f}$ relation based on shearable obstacles, or $\Delta\tau \sim$ f [82] based on short range ordering contributions).

3.2.4 Positron annihilation spectroscopy

Positron annihilation lifetime spectroscopy (PALS) and Doppler broadening (DB) can be used to study clustering phenomena. Positron lifetime is sensitive to electron densities around annihilation sites in the material; earlier, PALS has been used for measuring vacancy concentrations in metals. In principle, DB can be used to study the chemical environment of trapping sites. For positrons different annihilation sites exist in the material, but mixtures of different lifetime signals can be difficult to separate in PALS. [4, 13] Clustering during NA phenomena in AlMgSi alloys has been studied via PALS [87]. A schematic signal for the positron lifetime (τ) is shown in FIG. 3.20 [4]. The NA process has been subdivided into 5 stages by Banhart et al. [4] according to the curve found (where stage 0 is not observable). The first stage is characterized by a constant lifetime for Si-excess, followed by a decrease to a minimum at about 60 to 80 min, followed by a re-increase (stage 3). After reaching

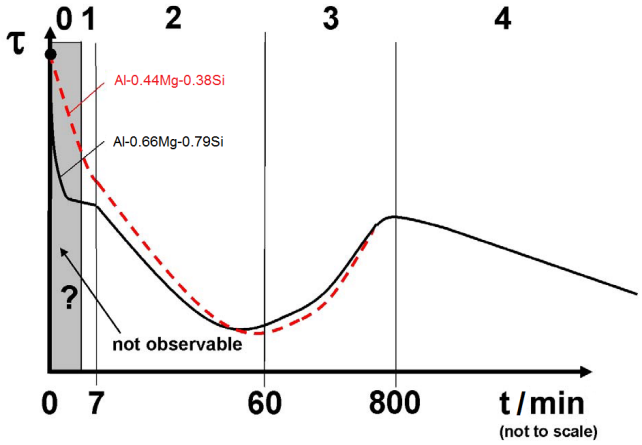

Figure 3.20: Schematic course of the average positron lifetime over natural aging time after solution heat treatment for Al-0.44%Mg-0.38%Si (at.) and Al-0.66%Mg-0.79%Si alloy. Black dot corresponds to the lifetime of free vacancies in Al (0.25 ns) which is considered the starting point of the curve. Adapted from [4].

a local maximum at approximately 800 min the lifetime decreased slowly until >104 min (stage 4). For a balanced alloy stage 1 is different [4]. Stage 2 was investigated in detail for a balanced low content alloy; the measured signal was seen to be influenced by the sort of quenching (FIG. 3.21) [88]. The measuring signal was seen to be dependent on the temperature during the measurement, which generally caused parallel shifts of the curves to lower lifetimes for lower temperatures [88]. The quench sensitivity was also studied by Strobel et. al. [53]. They investigated a balanced low content alloy (AA 6060) and found the above mentioned type of curve in their measurements of a water-quenched (WQ) sample. However, the overall picture changes for slower cooling rates, as shown in FIG. 3.22 [53]. Increasing lifetimes are seen until the end of stage 2 of the oil-quenched sample in [53]. Similar behavior was seen for NA at elevated RT (37 °C) in [87], but with lower absolute increase. The effect of additional elements has also been investigated. For example, adding Cu causes a concentration dependent time-shift to longer times of the local minimum (end stage 2) and seems to reduce the time from local minimum to local maximum [48]. This corresponds to a delay in hardness increase for Cu alloyed samples early in time and the outpace later in time of the non-Cu added alloy [48]. Such behavior has not been seen in pure Al 99.99 % during RT storage after quenching [53]. The authors show that the lifetime decreases monotonously (FIG. 3.23 [53]) and that the lifetime for WQ material is reasonable higher after 104 min than for Al nearly free from quenched vacancies. The lifetime signal for a pure Al is interpreted that vacancies (as positron traps) concentration decreases over RT time. For AlMgSi alloys early vacancy concentrations decrease (decreasing signal) and superposition the signal from forming clusters. They also act as positron traps and finally dominate the signal contribution (increasing lifetime). The again falling lifetime for long NA

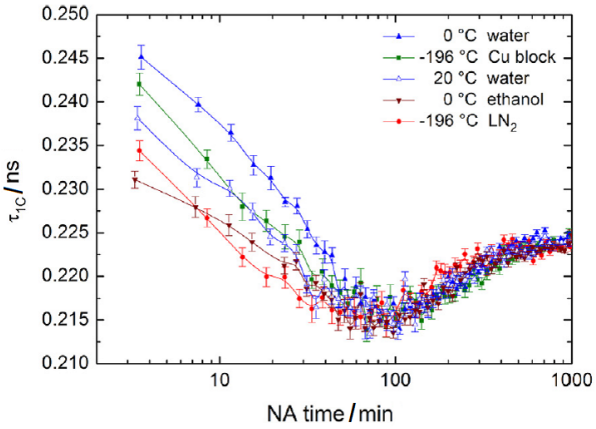

Figure 3.21: One-component positron lifetime in samples of alloy Al-0.4%Mg-0.4%Si after quenching into various quenching media. Adapted with permission from [88], ©2018 Elsevier.

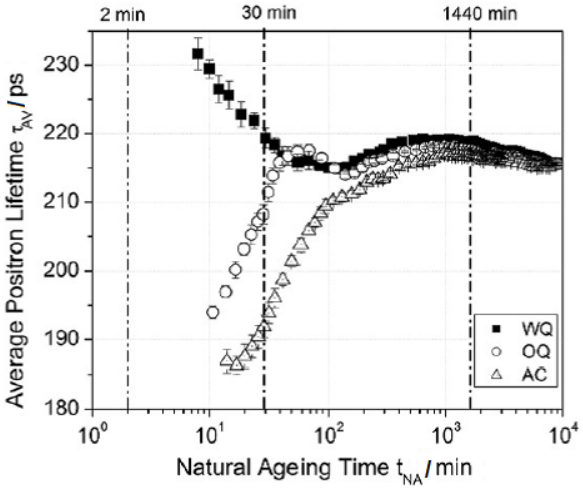

Figure 3.22: Evolution of average positron lifetime during natural aging of AA 6060 for different quenching conditions. WQ – water quenched, OQ – oil quenched and AC – air cooled. Adapted with permission from [53], ©2018 Elsevier.

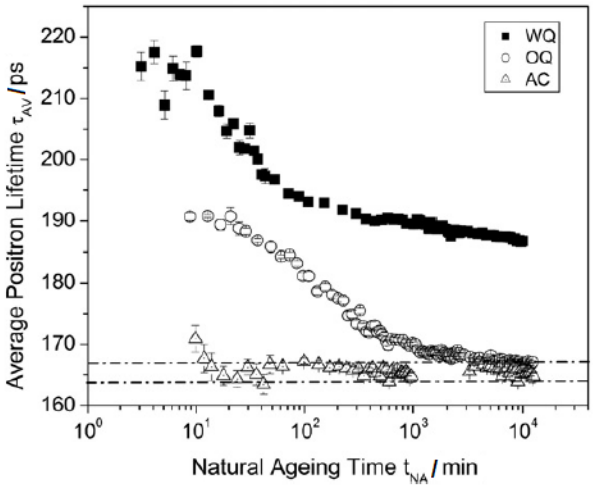

Figure 3.23: Evolution of average positron lifetime during natural aging of pure Al (99.99%) for different quenching conditions. WQ – water quenched, OQ – oil quenched and AC – air cooled. The horizontal lines indicate the average positron lifetime for the well-annealed and slow cooled Al in literature. [53] Adapted with permission from [53], ©2018 Elsevier.

times is somehow surprising, but could be interpreted by a change in chemistry /ordering of clusters.

3.2.5 Other techniques

Other techniques are sometimes used to characterize cluster evolution in aluminum alloys. Note that for the 2000 [32, 89] and 7000 series [90], in-situ small angle x-ray scattering (SAXS), and in [32] SAXS with nuclear magnetic resonance (NMR), was used to characterize clusters. However, these techniques cannot be deployed in a satisfying manner for the important 6000 alloys [4]. Small angle neutron scattering (SANS) has been applied for AlMgSi, but generated insufficient results [4]. Time dependent magnetization [91] and muon measurements have also been used recently [92, 93].

3.3 Direct characterization of clusters - atom probe tomography

In contrast to most microscopic imaging techniques, for example scanning electron microscopy (SEM) or TEM, atom probe tomography measures direct particle properties of ions of the target material, compared to secondary signals resulting from impinging waves in other methods. This makes it a destructive technique. Nowadays it is a frequently used tool, especially in materials sciences due to its strength in content analysis at the nanometer scale of

precipitates, grain boundaries, nano-particles, clusters and the matrix compositions in three dimensions. [94–96]. The technique is unique for visualizing and measuring of fine-scaled microstructural (nanostructural) features with near atomic resolution and gaining precise information of compositions in 3D. Especially, where transmission electron microscopy does not produce distinct contrast (e.g. for low alloying element content and elements of similar atomic number, which is the case for AlMgSi alloys [4, 97]), APT is the sole technique for gaining direct insight into the early decomposition of metallic systems. [56]

3.3.1 Functionality

A short overview is given over the principal functionality of an atom probe in the following text. A needle-shaped sample is fabricated, with a tip radius in the range of 50 nm. The most common techniques to achieve this is either by two-step electro-polishing [98] or sample preparation via a focused ion beam [96]. In an ultra-high vacuum chamber a high positive voltage (DC or standing voltage) is applied to the specimen, creating a high field environment at its apex; an additional impulse signal, provided by a voltage or laser pulse is applied at a high repetition rate (typically between 100-500kHz). Nowadays, atom probes with a local electrode setup are usually used. The local electrode is cone shaped with an aperture at its center; it is positioned in front of the specimen apex along the needle axis. In voltage mode the pulsed voltage is applied on the local electrode (with a negative bias), thereby causing a momentary increase in electric field. This setup enables a much larger field of view than earlier setups, and additionally has instrumental advantages. [96] The combined DC and pulsed high voltage yields to high electrical fields at the specimens' apex and, when a critical field is reached, a surface atom of the specimen is ionized in a process called field evaporation. Tabulated evaporation fields values exist for each element (usually calculated via a simplified model [94]), however the exact physical and electronic processes are still under debate in the community. There are various relationships between analysis parameters and it should be noted that the field evaporation process is temperature dependent such that at higher temperatures generally lower fields are required for field evaporation. The resulting ion is ejected from the sample and accelerated through the aperture in the local electrode by the local electric field, towards a two-dimensional position sensitive detector. To measure the field-evaporated ions in a controlled manner the pulsed signal is used as a start clock to a time-of-flight (ToF) mass spectrometer. The arrival time and two spatial coordinates are then recorded by the detector. The resulting ToF is used to calculate the mass-to-charge state ratio (typically measured in Da), which is then assigned to the species. This information, combined with the sequence of the arriving ions [99], is used to build a three dimensional reconstruction – the so-called atom map, ultimately representing relative positions of the atoms originating from the specimen apex [100]. [95] Multiple parameters define the 'parameter space' for collecting a successful atom probe tomography experiment;

the most important ones include: voltage pulse fraction or laser pulse energy, specimen temperature, detection rate, and pulse frequency. [96]

3.3.2 Artefacts, trajectories and calibration

Besides its strengths, atom probe tomography also has its drawbacks, as with any characterization method. In general, the resulting analyzed volumes of material compared to other techniques are small (maximal spatial extensions up to hundreds of nanometers). A further aspect is that not all atoms are detected. The two dimensional detectors have usually a detection efficiency of ∼37 to ∼57 %, though most recent developments have improved detectors (as in the Cameca LEAP 5000) up to 80 % [101]. Therefore, only a fraction of the atoms of the original sample are pictured in the reconstruction. This is especially crucial for small sized features such as clusters. With high enough detection efficiency cluster size distributions can be estimated to real clusters size distributions [102]. Another limitation is the loss of crystallographic information, strongly dependent on the alloy and – experimental parameters; in some regions of the reconstruction can lattice planes of certain directions be resolved (pole regions – poles). Although, in special cases, APT can be used to clarify occupancy of elements on sub-lattices [103, 104] or it has been shown that crystallographic arrangement of atoms may be regained altogether [105, 106]. Pole regions are visible due to crystallographic faceting of the sample during the experiment; the field-evaporated ions near to a pole are deflected away locally from the flat regions, which leads to lower density of pole regions in the reconstruction. This is a so-called trajectory aberration since the trajectory of the ion is influenced by its local neighborhood on the specimen surface. A similar artefact exists for precipitates which have a different evaporation field (high-field, low-field) compared to the matrix of the sample; this leads to precipitates appearing less dense, respectively denser in the final reconstruction. These are known as local magnification (demagnification) artifacts. [95] Crystallographic pole regions are often present when Al alloys are investigated with APT. With the use of spatial distribution maps (SDM) [107] or other techniques where the inter-planar lattice spacings can be measured, the reconstruction can be calibrated. Often the protocol based on a modified stereographic projection is used for the reconstruction [108], here the two parameters (namely the image compression factor and the field factor times the evaporation field) can be used to tune the reconstruction, so that the inter-planar distances in the atom map fit to the real inter-planar distances [109, 110]. It should be noted that there are large and ongoing efforts being made to increase the accuracy of the reconstruction algorithm [100], recently also revisiting a different projection model [111] or building the reconstruction in a reverse manner and calculating the ion trajectories [112]. Distorted or bad reconstructions would affect cluster analysis in shape or absolute size analysis, and worsen comparability in between different measurements. In general, different elements have different evaporation fields. In alloys the evaporation field of an element can be different from

the evaporation field of the pure metal since the evaporation field depends on the chemical (local bonding) environment, this is especially true if the species is bound in an intermetallic phase [94]. However, if the evaporation field of solute elements differ largely from the matrix evaporation field, preferential retention (for solutes with a higher evaporation field) can occur. To hinder this an adequate choice of measurement parameter space is important, chiefly among them being specimen temperature and pulse fraction [95]. Preferential retention of one species can lead to surface migration during the APT experiment. Surface migration is possible to occur for interstitial and substitutional elements [113]. A prominent example in Al alloys is Si, which is known to migrate to the (111)-pole [114]. The migration of solutes during the experiment results in false reconstructed location, and thus errors introduced in elemental distributions. To minimize this artefact optimized parameter spaces are to be applied, and if still present in the reconstruction, usually the distinct regions affected by this known artifact can be removed – though one should keep in mind that this potentially could affect the analysis, since the starting positions of the migrated atoms are unknown. A similar effect, but for precipitates, is seen for example in Al-Ag alloys, where the Ag atoms in the precipitates tend to be shifted to the nearest low-index pole [115]. It is concluded that subtle changes of the evaporation field of the matrix atoms in the precipitates cause this aberration, by changing the local field distribution. This is known to occur in various systems. Due to the aberrations specific to certain atomic species this is called chromatic aberration. [95, 115] In general, it is not straight forward to characterize clusters even with APT, due to the above mentioned challenges. In recent literature these issues are gaining consideration more and related topics are continuously being discussed in the community. However, this has not always been the case and full evaluations of these influences on the analysis of clusters is usually not given. Although one needs to be aware of these artifacts APT is still the most powerful technique to gain insight into clustering.

3.3.3 Analysis of solute distribution

There are many approaches to analyze the solute distributions that can occur within APT datasets and many can consider fine-grained and possibly subtle variation in composition. Two of interest to the analysis of fine solute clustering are cluster-finding algorithms and pair correlation algorithms.

3.3.3.1 Clusterfinding and algorithms

The most frequently used cluster-finding algorithm for APT analysis of aluminum alloys was named "maximum separation" [116], but in other fields it has been known as "single linkage" and "friends-of-friends". For APT data, maximum separation connects data points from a particular solute range, and two solute atoms are connected if they are closer than a certain distance $d_{\max}$, this parameter chosen by the user. Choosing a $d_{\max}$ value filters the solute

data based upon its first nearest neighbor (1NN) distance distribution. Better discrimination between physical clusters and the adjacent matrix, though with a lower sensitivity to smaller clusters, can be achieved through applying this filtering upon the kth nearest neighbor distance distribution. In other words, a population of core atoms are identified by testing the expression $d_{\max} < d_{\mathrm{kNN}}$ for each solute atom. To find clusters, these core atoms are then linked with any solute atoms (both core & filtered) that are closer than $d_{\max}$. This is the basis of the DBSCAN algorithm [117]. Further filtering can be applied by removing found groups smaller than $\mathrm{N_{min}}$ detected solute atoms, and for a chemical analysis, enveloping and erosion steps were then used to include non-solute atoms [118]. The "maximum separation" technique has long been implemented in the commercial IVAS$^{\mathrm{TM}}$ software by Cameca Instruments Ltd. as "envelope". The DBSCAN algorithm (of which the maximum separation algorithm is a special case where k = 1) is separately implemented in IVAS$^{\mathrm{TM}}$ under the option "Create Cluster Analysis". Results can be sensitive to the choice of algorithm and parameter values. Results are also dependent upon the studied material and the capabilities of a particular atom probe instrument. In other words, these methods cannot be taken from a particular study and applied to another without considerable thought. This review study confines itself on the use of the maximum separation and DBSCAN algorithms for their ubiquitous use. However, we must acknowledge that these algorithms may not be optimal for all purposes. There are many other algorithms that have been developed and considered for APT data analysis, some similar to maximum separation and DBSCAN [119–121], some employing mixture models [122] and others dependent upon computational or networking geometry [123–125]. This should not be regarded as an exhaustive list and there are many more algorithms besides which have not yet been applied to APT data analysis. Justifying "good" parameter selection is often an annoyingly difficult task and, in many respects, it must be remembered that it is merely ancillary to the materials science problem. Rigorous approaches to using the maximum separation algorithm have been developed; exploring the sensitivity of maximum separation [126], examining a manifold of results with the variation of parameters [127], and using simulations to determine optimal parameters [128]. It can be hoped that unsupervised machine learning approaches can be developed and trained in combination with APT data simulations. Heuristic rules have also been used to establish consistent cluster-finding analyses (but not necessarily providing optimal or correct results) and many of these have been considered in 3.2. Selecting parameters based upon establishing the least conservative parameter values that resulted in no "random" clusters being found in a random labelled version of the original APT data [64, 101, 129–134]. Other more elaborate approaches using randomized datasets to generate various heuristic metrics have been developed [135–137] but could be considered as less than physical so caution must be employed to at least apply these consistently.

3.3.3.2 Interpretation via pair correlation functions/ partial RDF

Pair correlation and partial radial distribution functions provide the means to directly investigate solute-solute interactions on short- to long-range orders in aluminum alloys. Pair correlation functions were developed for such analyses without recourse to the complexities behind cluster-finding algorithms [138, 139]. The similar application of radial distribution functions was investigated for bulk metallic glasses [140]. APT data can provide the means to directly query short range order; for multicomponent alloys, this has been recently formalized [141].

3.3.4 Findings on clustering in aluminum alloys

Over time different atom probe instruments applying varying APT measurement parameters and sample preparation methods have been used. Moreover there are differences in the solution heat treatment and quenching which is the basis for the studied SSSS. This information on measurements needs to be considered and is comprised in 3.1 for all referred studies. Since often also different parameters for the clustering algorithm are used, this information is summarized in 3.2. Additionally, the investigated aluminum alloys are summarized in view of content in 3.3. In terms of detailed results on clusters we focus here on the important class of AlMgSi alloys.

Table 3.1: Overview over the experimental parameters for studies discussed in direct characterization of clusters. APT - the type of the atom probe used, PF – pulse fraction, T – Specimen temperature, SHT – solution heat treatment temperature (T) and time (t), Quenching – type of quenching from SHT temperature used, Ref. – Reference.

APT	PF / %	T / K	Vacuum	Preparation Method	SHT T/°C / t/min	Quenching	Ref.
APFIM	20	25	$< 1.5\times10^{-10}$ mbar	N/A	530 / 90	Ice water	[7]
1DAP & 3DAP	20	30	1×10^{-10} Torr	N/A	550 / 30	Water	[6]
ECoTAP	20	40 / 80	N/A	5% $HClO_4$ in 2-butoxyethanol	527 / 30	Water	[138]
PPoSAP	20	30	$< 10^{-10}$ mbar	Standard two-step	560 / 30	Water	[64]
Imago 3DAP	20	30	$< 10^{-8}$ Pa	N/A	560 / 30	Ice water	[73]
3DAP ON3DAP	20	25	$\sim3.4\times10^{-11}$	N/A	550 / 30	Water	[137]
LEAP 3000X Si	20	∼20		Standard two-step	560 / 10	Water	[135]
Imago 3DAP	20	30	$< 10^{-8}$	N/A	560 / 30	Ice water	[142]
LEAP 3000 X HR	15	35	$< 10^{-10}$ mbar	Standard two-step	570 / 20	N/A	[143]
Energy compensated	20	40	$< 10^{-8}$ Pa	25% $HClO_4$ in acetic acid 1st & 2 % $HClO_4$ in 2-buthoxyethanol	540 / 60	Water	[136]
Reflectron ON3DAP	20	25	3.4×10^{-11} mbar	N/A	550 / 30	Water	[62]
LEAP 4000 X HR	20	23.7	$< 10^{-10}$ mbar	Standard two-step	570 / 20	Water	[44]

LEAP 3000 X HR	15	35	$< 10^{-10}$ mbar	Standard two-step (15 % $HClO_4$ in acetic acid & 2% $HClO_4$ in 2-buthoxyethanol)	570 / 20	Water 900°C/s	[12]
LEAP and LAR3DAP	15-20	25-30	N/A	Standard electro-polishing	562 / 30	Water	[129, 130, 133]
LEAP 4000 HR	20	25	$< 10^{-10}$ mbar	Standard two-stage	560 / 30	Water	[132]
LEAP 3000 HR	20	30	$< 10^{-8}$ Pa	Standard two-step (25 % $HClO_4$ in acetic acid & 2% $HClO_4$ in 2-buthoxyethanol)	570 / 30	Water	[131]
LEAP 3000 HR	20	30	$< 10^{-8}$ Pa	Standard two-step (25 % $HClO_4$ in acetic acid & 2% $HClO_4$ in 2-buthoxyethanol)	570 / 30	Water 90°C	[134]
LEAP 4000 HR	20	33	N/A	Cryo-FIB	530 / 5	LN2	[144]
LEAP 3000 HR & LEAP 5000 XS	20	30	1.0×10^{-8} Pa	Standard two-step	570 / 30	N/A	[101]
LEAP 4000 HR	20	20	$<5\times10^{-11}$ Torr	10% perchloric acid in glacial acetic acid, 2% perchloric acid in butoxyethanol	560 / 75	N/A	[145]
LEAP 3000 HR	20	30	$<10^{-8}$Pa	Standard electro-polishing	560 / 30	Water	[146]

3.3.5 Important early findings

An early and famous reference investigation of clustering and precipitation sequence of the AlMgSi system via an atom probe like technique (atom probe field ion microscopy APFIM) was done by Edwards et al. in 1998 [7]. The studied alloy was AA 6061 and the results

are interpreted in terms of concentration profiles along cylinders of approximately 1.6 nm in diameter. For a heating of an as-quenched alloy with 5 K/min to 100 °C Mg, Si and Mg-Si co-clusters were detected. Longer times (8 h and 60 h) of aging at 70 °C showed existing Mg-Si co-clusters, for 0.5 h of aging the data was not conclusive and did not show Mg-Si co-clusters via contingency table testing. Here the data is grouped into blocks and the number of blocks containing different amounts of specified atoms are counted and can be compared and tested against randomized data [95]. The Mg/Si ratio for most of the clusters found for 8 h aging was 0.7 and strongly deviating in both directions. For 60 h of aging at 70 °C the ratio of most of the clusters was close to unity, which is close to the Mg/Si ratio in the β'' precipitates. Interestingly these finding already fit well into the recent picture of early clustering, which was developed during the subsequent 20 years (see section 4.5). Cu has not been found enriched in the clusters. It was concluded that independent Si and Mg clusters do form first followed by the formation of Mg-Si co-clusters. It was speculated that either or both Mg and Si clusters formed directly after quenching. It is already outlined that measuring the as-quenched state of the alloy is difficult due to preparation time needed at RT after quenching. Finally the following initial cluster sequence based upon previous thermal analysis and atom probe analysis was suggested [7]:

$$\text{SSSS} \rightarrow \text{clusters of Si and clusters of Mg} \rightarrow \text{wdissolution of Mg clusters} \rightarrow \text{Mg/Si co-clusters} \tag{3.5}$$

Note that, although this was a reference for numerous studies over last two decades, the dissolution of Mg cluster in the sequence has not been confirmed and seems to be questionable. Another important early APT work by Murayama et. al. [6] investigated the interaction for a balanced and a Mg excess AlMgSi alloy without Cu. The balanced material was studied in detail as described in the following text if not otherwise mentioned. Only a uniform fringe contrast for long-term NA could be obtained in high resolution transmission electron microscope (HRTEM), in comparison to PA (here 16 h at 70 °C) where approx. 2 nm sized clusters (arbitrarily termed as GP zones) are observable. Contingency tables from APT data indicate positive correlation for Mg and Si for long-term NA, although no visual appearance can be found. Integral profiles of Si and Mg show the presence of Mg, Si and Mg-Si clusters, and the Mg-Si cluster have a Mg/Si-ratio close to unity. Strong correlation in between Mg and Si is found for PA where enriched regions can also be observed by eye. Mg-Si co-clusters show a Mg/Si-ratio of approximately unity. Also the interaction of different types of clusters created via PA and NA on the formation of β'' precipitates was studied. It was concluded that NA clusters do not act as nucleation sites for β'', while PA clusters act as nuclei [6]. Moreover, the authors summarize that in the as-quenched condition separate Si and Mg clusters exist and Mg and Si clusters aggregate during NA. The different effect of co-clusters formed upon NA and PA was argued based on a critical radius for nucleation of β'' precipitates, so that small co-clusters (NA) revert at AA temperatures. It is also already

stated that co-clusters, GP zones and β'' precipitates follow the overall alloy composition. [6]

3.3.6 Latest findings

Since these early works, numerous studies have been carried out. As time progressed, the atom probes evolved and the data analysis methods got more powerful and extensive [147]. Even the earlier used analysis tools such as the 1D concentration profiles could be misleading due to random statistical fluctuations which could be wrongly interpreted as clusters [147, 148]. Today much larger data sizes are usually gained in the experiment, but due to the small size of clusters, APT is still operating at its limits for this purpose. In later works the effect of clusters on further different heat treatment procedures is investigated. The evolution of clusters during RT storage and PA, influence of combined heat treatments as NA and AA, or PA and AA is studied. Also the influence of alloy chemistry on the clustering behavior is analyzed. The findings have been grouped by the different applied heat treatment states and are discussed in the following.

3.3.6.1 Natural aging

Only recently, it was shown that the as-quenched condition for an excess Si alloy ($\sim$1 min of RT storage) can be measured via APT [144]. Natural aging times $<$ 60 min became available due to the use of a customized cryo-transfer system to the APT measurement chamber [149, 150]. No Si-Si, Mg-Mg nor Mg-Si correlation, with a radial distribution function - like measure, is seen. [144] Based on a 5th nearest neighbor distribution it is shown that no clusters are expected for the Cu-containing alloy AA 6111 for 2 h of NA. [135] In [142] it is stated that also for an Mg excess alloy for the quenched state (time at RT for APT sample preparation and transfer into the APT analysis chamber is not given in [142] but is technologically expected to be $>$ 1 h for the used setup), no clusters were found. However, these data [135, 142] contradict the results of other studies. Contrary, in [138] for the alloy 6016 and 60 min natural aging the material already shows positive autocorrelation for Mg-Mg and Si-Si, but not for Si-Mg. Therefore, it was concluded that two distinct populations of clusters form for short RT storage, as already stated in the sequences 3.5. A large proportion of Mg is involved in the clusters as compared to Si, where the proportion in solution is higher. For one week at RT only significant Si-Si correlation was found, which would actually support sequence 3.5 where early Mg-clusters again revert. [138] This finding is often contradicted by other studies where usually Mg, Si co-clusters are found, although radial distribution functions as a measure of clustering in literature is rare [105, 138, 144] and therefore direct comparison to other measures is difficult.

Table 3.2: Overview over cluster search algorithm parameters used for cluster identification (same cases as in 3.1). d_{max} – maximum separation distance, N_{min} / K – minimum nearest neighbor for a cluster / nearest neighbor for core identification, L – linking distance, d_{erode} – erosion distance, Core atoms – atoms chosen as possible core atoms in the cluster identification algorithm, Parameter choice – on which basis the parameters are chosen, Ref. – Reference.

d_{max} / nm	N_{min} / K	L / nm	d_{erode} /nm	Core atoms	Parameter choice	Ref.
0.70-0.80	10	N/A	N/A	Mg, Si, Cu	No random clusters identified	[64]
0.60	20	N/A	N/A	Mg, Si	N/A	[73]
0.62	6	d_{max}	d_{max}	Mg, Si, Cu	Max. ratio actual to random signals	[137]
See parameter choice	2 / 5	$d_{max}/2$	N/A	Mg, Si, Cu	Max sum(5NN)-sum($5NN_{rand}$)	[135]
0.45	6	N/A	N/A	Mg, Si, Cu, Ag	N/A	[142]
0.48	N/A	0.48	0.48	Mg, Si, Cu	[118]	[143]
0.60	8	N/A	N/A	Mg, Si	Low noise-per-cluster	[136]
0.50	4	d_{max}	L	Mg, Si, Cu	Literature and atomic reconstruction results [151]	[62]
0.65	10 / 5	N/A	N/A	Mg, Si	No aggregates found in random comparators	LEAP [129, 130]
0.70	10 / 5	N/A	N/A	Mg, Si	No aggregates found in random comparators	LAR3DAP [129, 130]
0.60 / 0.65	10 / 5	N/A	N/A	Mg, Si, Cu	No aggregates found in random comparators	LEAP [133]
0.50	10	N/A	N/A	Solute atoms	No aggregates found in random solid solution [128]	[132]
0.75	10	N/A	N/A	Mg, Si	No aggregates found in random solid solution	[131]
0.75	10	N/A	N/A	Mg, Si	No aggregates found in random solid solution	[134]

0.74	5 / 5	$d_{\max}$	$d_{\max}$	Mg, Si, Cu	N/A	[144]
0.75 / 0.65	10 / 20	N/A	N/A	Mg, Si	No aggregates found in random solid solution	[101]
See parameter choice	5 and 1 / 65 and	1/2 $d_{\max}$	1/2 $d_{\max}$	Mg, Si	Max sum(5NN)-sum($5NN_{rand}$) / Max sum(1NN)-sum($1NN_{rand}$)	[145]
0.75	10	N/A	N/A	Mg, Si	No aggregates found in random solid solution	[146]

Cao et al. reported that after 1.1 h of NA, Mg and Si show correlation in contingency table analysis for an excess Si alloy, whereas Mg-Cu and Si-Cu show no significant difference from a random solution. With NA time the significance is increased for any of the two combinations of Mg, Si and Cu. Cu is therefore concluded to cluster after slightly higher NA times ($\geq$ 3 h). The number density of identified clusters stagnates after 24h of NA. [62] For AA 6111, increasing number densities within up to two weeks of Mg-Si clusters are reported in [135]. In general, the found clusters have a size expressed in solute atom numbers $\leq$ 50 (here the number of solutes in the APT datasets are given, not the physical values, these values should be corrected for detection efficiency of the specific atom probe, 3.1). Mg-Si clusters dominate the cluster population, Mg-Si-Cu, Cu-Mg and Mg-Mg clusters roughly follow this scheme of increasing number densities. About 10 to 20 % of the solutes are reported to be bound in aggregates for NA samples. Also only small changes in cluster size is reported for NA. The volume fraction of atoms in aggregates is seen to increase over the first week of NA (which is correlating with yield strength). They also observe a decrease of the number densities of clusters after one week of NA, which is however not explained and seems to be physically unrealistic. [135] For increasing NA time starting from 3 h to 325 days Aruga et al. obtained increasing number density in a balanced alloy in [131]. The Mg/Si ratio of 0.8-1.0 had the highest number density in all NA states. The number density of Si-rich clusters is shown to stagnate already after short NA times. It is concluded that Si-rich clusters can form at the earlier stage of NA. Moreover, Aruga et al. [131] suggest that non-solvable Si-rich clusters may reduce supersaturation and lead to a retardation of the hardness increase during AA. No influence of the size of clusters on reversion was seen, contradicting the conclusions of Murayama et al. [6]. [131] For Si-rich alloys the cluster number density is shown by Jia et al. [132] to be higher, in comparison to Mg-rich alloys, after two weeks of NA. The average composition of the clusters is reported to follow the alloy composition. [132] Recently, Zandbergen et al. show a substantial number of clusters to be formed within 100 min of NA which then only slightly increase in size for further NA, but the number density increases up to one week of NA [129, 130]. The clusters contain mainly

Mg and Si, 2% of them are only-Si-clusters, but no Mg-only clusters have been detected at any stage [129, 130]. It is concluded that limited diffusion capabilities drives the clusters at NA towards a "metastable state from which it is difficult to escape either energetically or kinetically". [129] Zandbergen et al. [133] also reported that only small differences are obtained for NA of AlMgSi alloys with the addition of Cu in cluster numbers and sizes. The most important difference is obtained in the composition, the majority of clusters do contain Cu (high-Cu content alloy). The Mg/Si ratio is increased for the high-Cu alloy, also the (Mg+Cu)/Si ratio is increased, here to about unity. [133] Jia et. al [132] also show that for Cu added alloys the larger clusters found approached a Mg/Si ratio of 1.0. Furthermore, Cu is suggested to change the stability of Mg-Si co-clusters through its incorporation and enables them to transform more easily to hardening phases upon AA. [132] Trace elements have been reported to suppress clustering during NA via APT. For two weeks of NA of AA 6061 alloy with the addition of Sn, a 1st NN spatial distribution shows no discernable difference from a random distribution of Mg and Si atoms. [44] We conclude the following: No clusters are found in the as-quenched state Mg, Si and or Mg, Si co-clusters are detected after a very short time of NA (60 – 100 min) by several studies, but there are also contradictions Mg, Si co-clusters have been always found after long-term NA Mg, Si co-clusters increase small in size for short times, then stagnate in size Si-rich clusters number densities already stagnate after short NA times Increasing number densities of clusters up to one year of NA are confirmed by several studies, but there are also contradictions Cu aggregates at later stage of NA to the clusters Only small difference in size and number density for NA clusters for Cu containing alloys are observed, but the Mg/Si- ratio is increased and the majority does contain Cu. In general, the very early stages of NA seem to be still unclear because of contradicting APT results. This may be caused by reaching the limits of detection, APT artefacts and an uncompleted selection of the investigated alloys.

Table 3.3: Overview of the AlMgSi alloys and aging treatments investigated by APT (same cases as in 3.1 and 3.2), Mg, Si and Cu contents. If wt.% were originally given, the at.% values are calculated on the assumption that only Al, Mg, Si and Cu exist in the alloy. Cu contents are set to zero if not explicitly given in the reference. Mg/(Mg+Si) and Mg+Si values are calculated from this contents.

Alloy #	Mg / at.-%	Si / at.-%	Cu / at.-%	Mg/(Mg+Si)	Mg+Si	Aging	Ref.
6061	0.89	0.76	0.08	0.54	1.65	PA, AA	[7]
N/A	0.70	0.33	0.00	0.68	1.33	NA, PA	[6]
	0.65	0.7	0.00	0.48	1.35		
6016	0.44	0.96	0.00	0.32	1.41	NA, PA, AA, PA+AA	[138]
6111	0.90	0.60	0.30	0.60	1.50	PA, AA	[64]
N/A	1.05	0.78	0.00	0.57	1.83	NA, PA, BH, NA+BH, PA+NA, PA+NA+BH	[73]
6182A 6022-type	0.97	0.83	0.01	0.54	1.80	PA+NA, NA, NA+AA, PA+NA +AA	[137]
	0.54	1.03	0.12	0.34	1.57		
6111	0.90	0.60	0.30	0.60	1.50	NA, PA, AA	[135]
N/A	1.01	0.78	None /Cu / Ag	0.57	1.78	NA, PA	[142]
6061	0.92	0.58	0.09	0.61	1.50	AA T6, NA+AA T6	[143]
N/A	0.86	0.43	0.00	0.67	1.29	NA+PA, PA	[136]
	0.40	0.84	0.00	0.32	1.24		
N/A	0.53	1.03	0.12	0.34	1.56	NA, NA+BH	[62]
6061 + Sn	0.90	0.59	0.09	0.60	1.49	NA, AA	[44]
6061	0.92	0.58	0.09	0.61	1.5	NA, NA+AA, IQ to AA	[12]

N/A	0.51		0.94	0.01		0.35	1.45		AA, NA, PA, PA+NA, NA+BH, PA+NA+BH, spike+ PA	[129, 130, 133]
	0.51		0.94	0.01 / 0.06 / 0.34		0.35	1.45			
N/A	0.69	-	0.55 - 1.1	0.03	-	0.38 - 0.7	1.77	-	NA, NA+BH, NA+AA	[132]
	1.31			0.21			1.88			
N/A	0.69		0.89	0.00		0.44	1.58		NA, NA+BH	[131]
N/A	0.69		0.89	0.00		0.44	1.58		PA+NA	[134]
6016	0.40		1.00	0.03		0.29	1.40		NA	[144]
N/A	0.69		0.89	0.00		0.44	1.58		NA, PA	[101]
N/A	0.36		0.31	0.00		0.54	0.66		NA, AA	[41]
N/A	0.33		1.13	0.00		0.23	1.47		NA, NA+AA	[145]
	1.12		0.39	0.00		0.74	1.51			
N/A	0.69		0.89	0.00		0.44	1.58		NA, NA+BH	[146]
	0.87		0.67	0.00		0.56	1.54			

3.3.6.2 Pre-aging

In the following we discriminate in between NA+PA and PA+NA, where NA+PA means NA followed by PA and for PA+NA the reverse sequence. Moreover, interrupted quenching treatments are considered as PA treatments here. In [138] it was reported for AA6016 that PA (90°C for 8 h) plus NA lead to Si-Mg pair correlation values > 1. Esmaeili et al. [64] also performed APT investigations for different PA temperatures ranging from 60 °C to 180 °C for the alloy AA 6111, while adjusting the time to be at a local electrical resistivity maximum. Generally, Mg-Si aggregates were obtained, where a majority also contains Cu. It was revealed that with increasing temperature the number density of small clusters was reduced and a second family of large clusters arise, also the average Mg/Si-ratio was seen to increase. [64] A difference between NA and PA was revealed by Serizawa et al. [73] for an Mg-excess alloy. Clusters do not grow during NA and the majority of the clusters has $N_{solutes} < 50$ (not corrected for detection efficiency). Clusters formed upon PA at 100 °C increase in size (see FIG. 3.24) for a time, recalculated from PA time to time at RT equating diffusion distances and assuming a Ahrrenius type form of the diffusion constant with an activation energy of 130 kJmol-1. They postulated that clusters formed during NA and PA are different types of clusters (C1 and C2 respectively). The clusters showed various Mg/Si-ratios, especially if the clusters are small. Larger clusters were found to have a narrow Mg/Si-ratio distribution (C2), approaching a Mg/Si ratio of 1.5 to 2.0 and are believed to

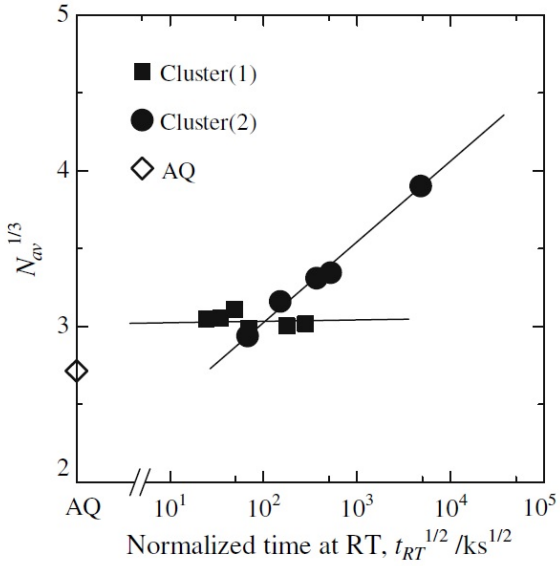

Figure 3.24: Growth rate of clusters formed at room temperature (Cluster (1)) and at 100°C (Cluster (2)). The values $N_{av}^{1/3}$ (N_{av} – average solute number of clusters, $N_{av}^{1/3} \sim$ average cluster radius) are plotted against the square root of the normalized room temperature time, $t_{RT}^{1/2}$ (t_{RT} – recalculated time, from time at PA temperature to time at room temperature RT), time starts after quenching (as-quenched AQ – here also the N_{min} value of the cluster search is printed as AQ, but not measured in AQ state). The average radius is proportional to the square root of time if the clusters grow with the diffusion mechanism. Adapted with permission from [73], ©2018 Springer Nature.

be able to transform into β'' precipitate upon AA. Serizawa et al. [73] also performed a comparison of PA and PA followed by NA (∼28 days), which showed similar Mg/Si-ratio over N_{Mg}+N_{Si} (number of solutes in a cluster, N_{Mg} number of Mg atoms and N_{Si} number of Si atoms in a clusters) plots. The Clusters (2) are therefore considered not to grow or dissolve at RT. [73] Torsæter et al. [136] reported in contrast that PA of Si-excess and Mg-excess alloys showed clusters of a Mg/Si ratio of approximately unity (broad peak of distribution around 1.0), which means that the composition of clusters formed upon PA does not depend on the alloy composition. This is in contrast to NA, where the average composition of NA clusters was concluded to follow more the overall composition of the alloy. If NA is examined prior to PA a large decrease in identified aggregates can be observed and a dual distribution of Mg/Si ratios is present. [136] Prolonged NA after PA (examined via quenching into heated water at 90°C) is investigated in [134]. NA after this PA resulted in an increase in number density and a decrease in average Mg/Si ratio of clusters. The number density of low Mg/Si ratio is seen to increase and cause the shift in average Mg/Si ratio. Larger Clusters are seen to have a higher Si content after PA+NA compared to PA only. It is believed that Si aggregates to the

clusters formed during PA rather than the independent formation of Si-based clusters. [134] Cao et al. [137] investigated AA 6181A and an AA 6022-type alloy. Natural aging of 24 h resulted in a high number density of small clusters in AA 6181A. PA (20 s at 200°C) plus NA 24 h yielded lower number density of small aggregates. Interestingly a strong correlation between the amount of large aggregates and yield strength was obtained, but the overall number density did correlate less with the yield strength. The alloy AA 6022 showed similar trends, but generally the fraction of solutes in the aggregates was lower and yielded lower mechanical properties. [137] In [41], it was shown that PA at 160 °C for 2 min (examined via interrupted quenching) can strongly suppress subsequent cluster formation during NA in a lean AlMgSi alloy. This was explained by a reduction of quenched-in vacancies rather than a formation of clusters upon this short term PA. Zandbergen et al. also confirmed that for approximately equal time spans of NA and PA, PA resulted in larger-sized clusters of higher Mg/Si ratio and the Mg/Si ratio distribution is narrower with a peak at unity. These clusters continue to grow upon further PA. Also it is reported that smaller clusters are fewer in the PA than in NA condition. For 10 hours of PA at 80 °C the number density stagnates for $N_{min} = 5$ in comparison to two hours PA at 80 °C, while for $N_{min} = 10$ the number density increases – which is interpreted as growth of clusters. Applying a spike (10 s 180°C) heat treatment after quenching prior to PA, the number density of larger clusters after PA is increased. Clusters formed upon PA are suggested to be similar in Mg/Si ratio like β'' precipitates and the average Mg/Si ratio is reported to be substantially lower for clusters formed during NA, which confirmed prior findings. [129, 130] Latest research compares the findings on NA and PA between the newly available high detection efficiency atom probe (LEAP 5000 XS) and atom probe data generated by lower detection efficiency (LEAP 3000 HR) for an excess Si alloy (see 3.3) [101]. Generally, a shift to larger cluster sizes, in the plot number density over Guinier radius, is obtained for a high detection efficiency atom probe. Also the average Mg/(Mg+Si) ratio is found slightly increased with higher detection efficiency. The atomic density of clusters is further investigated in detail. For clusters with a Mg/(Mg+Si) ≤ 0.4 an almost constant, significantly lower than average density is found. Therefore a ratio Mg/(Mg+Si) ≤ 0.4 is used to define Si-rich clusters. The volume fractions for short time NA, long term NA and PA for the so defined classes of clusters are analyzed. A significant lower volume fraction of Si-rich clusters for PA treated material is found in comparison to the NA treatments. Long term NA leads to higher volume fraction of Si-rich clusters, which are concluded to be the critical constituent for the occurrence of the negative effect. [101] We conclude: Clusters formed during NA and clusters formed during PA are possibly two different kinds of clusters PA forms Mg, Si co-clusters which act as nucleation site or can transform into β'' during AA PA clusters often show a Mg/Si-ratio of unity, while NA clusters show various Mg/Si-ratios, PA plus NA increases the number density of Si-rich clusters and decreases the average Mg/Si-ratio of the clusters NA leads to a significant amount of Si-rich clusters, whereas PA leads to larger volume fractions of Mg,

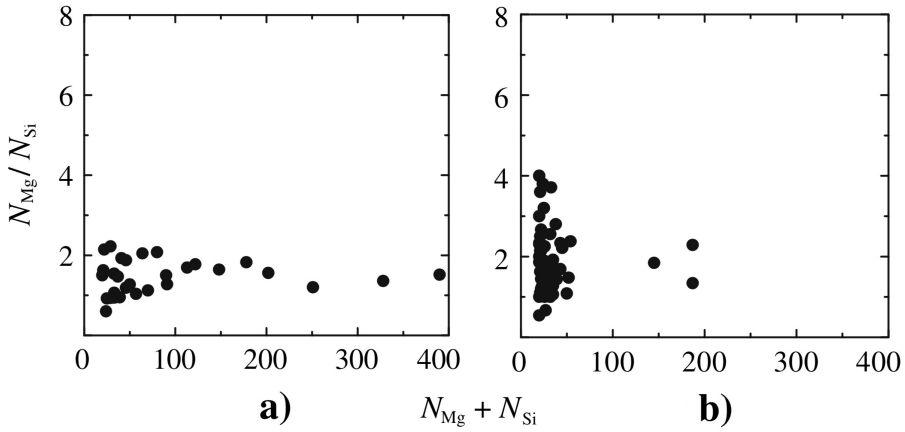

Figure 3.25: Relationship between size (N_{Mg}+ N_{Si} – number of solutes) and Mg/Si ratio of clusters and the β''phase for specimens (a) direct bake hardening treatment and (b) naturally aging for 604.8 ks and bake hardening. Reproduced with permission from [73], ©2018 Springer Nature.

Si co-clusters with a lower amount of Si-rich clusters In general, clusters formed upon PA are better understood than those formed by NA and can be clearly distinguished. The work of the last 20 years made clear that PA leads to clusters, which represent a pre-state of the precipitates formed upon AA. This is well resembled by the found composition spectrum.

3.3.6.3 Artificial aging

In this section APT results on short AA (called bake hardening (BH)) and AA to T6 illustrate the effect of clusters from NA and PA. A sample NA and a sample PA+NA are additionally artificially aged at 185 °C for 2 h and investigated in [138]. The PA sample showed higher number densities of aggregates and besides needle shaped β'' precipitates also spheroidal shaped ones in AA condition [138]. It is proposed that Mg-Si co-clusters act as nucleation sites for hardening phases [138]. Two hours of PA (80°C) was seen to not fully restore the BH response in comparison to direct AA, but increases it compared to naturally aged material [129]. The difference in short AA (20 min 170 °C) with and without NA is quantitatively shown for the identified clusters/precipitates in FIG. 3.25 [73]. A narrow Mg/Si-ratio distribution with lots of large precipitates is found for the direct BH in comparison to the NA+BH state, where a high number of small clusters is found with a large spread in Mg/Si-ratio and a second fraction of larger size but few in number. It is suggested that clusters from NA remains after BH. [73] This is confirmed in [12] where a decreasing density of clusters remaining from NA was observed upon AA. Already after 10 min of NA, the inhibiting nature of NA on BH is reported in [129]. The incompatibility of average Mg/Si ratio of NA clusters and precipitates is suggested as a mechanism of inhibition [129]. The difference in the T6 state upon longer AA with and without NA is illustrated in FIG. 3.26 for the alloy AA 6061 [143]. Artificial aging without NA resulted

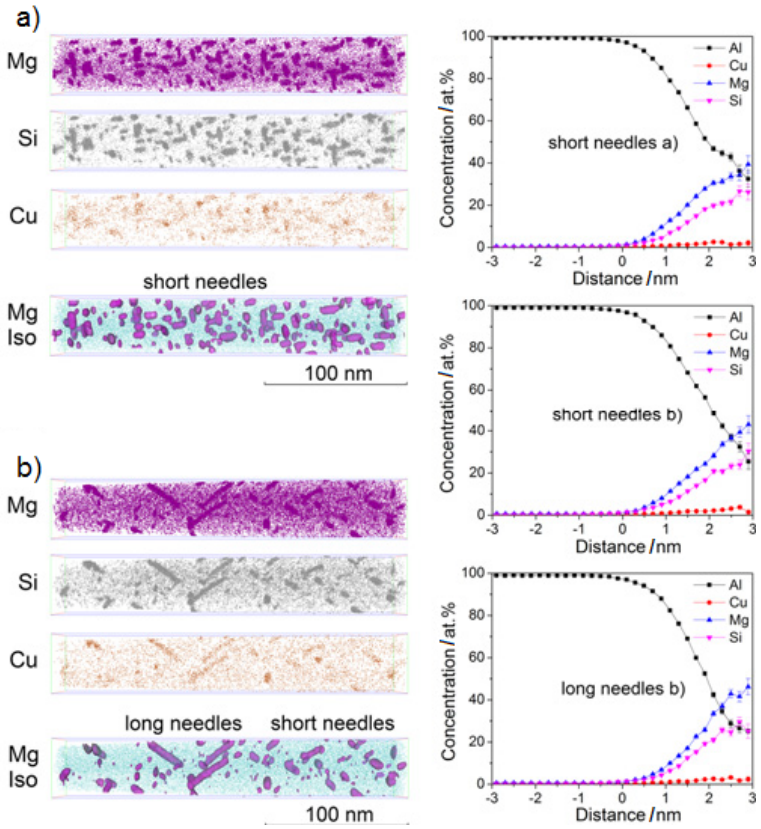

Figure 3.26: 3D reconstructions of atom positions for Mg, Si and Cu and isoconcentration surfaces of Mg embedded in the Al matrix with corresponding proximity histograms for Al, Mg, Si and Cu based on short needles and long needles for (a) direct artificial aging and (b) artificial aging after long-term natural aging (all in T6 condition). Adapted with permission from [143], ©2018 Elsevier.

in spheroidal precipitates and short needle-like β'' precipitates with high number density. For NA + AA clusters, spheroidal precipitates and long needle-like β'' precipitates with low number density are found. No significant compositional difference in between precipitates for NA+AA in comparison to direct AA was found. However a bimodal size distribution for β'' precipitates is obtained for the NA+AA heat treatment. [143] For NA+AA (170 °C 30 min), still a relatively high number of small solute aggregates is apparent, whereas the number of large aggregates decreases with increasing NA before AA (FIG. 3.10 b) [62]). The average size of clusters for NA+AA reaches a plateau after 3 h of NA. For NA only, average size increases slightly within the first few hours, but then remains constant (FIG. 3.10 a)). Comparing NA and NA+ AA, for longer NA times, average cluster sizes are the same, only the maximum cluster size is larger for NA+AA. [62] Aruga et al. [131] also found that the impact of increased NA after three hours of NA on the T6 state upon AA investigated via APT is low for an excess Si alloy, though with longer NA, a larger drop in hardness during short AA times is obtained. Poznak et. al. [145] conclude that the negative effect on AA is a

function of bulk Mg/Si ratio as stated by [152]. Further, they analyze cluster evolution during AA at 175°C for Si- and Mg-rich alloys. They conclude that NA clusters of the Mg-excess alloy are thermally stable at this low aging temperature and an irrecoverable negative effect is introduced as compared to the Si-rich alloy [145]. Similar findings are made by Aruga et. al. [146], where also a better recoverable (but not full in comparison to direct aging) for a Si-rich alloy in comparison to Mg-rich alloy for longer AA times is seen. Though, here for long NA plus BH a small increase in Si-rich clusters and a decrease for Mg-richer clusters is seen [146]. With higher Cu content Zandbergen et al. [133] reported that the direct BH response increases and a larger number density of short length precipitates is found. The (Mg+Cu)/Si ratio increases from the low- to high-Cu content alloy, while the Mg/Si ratio decreases. In increased Cu content alloys all precipitates contain Cu. It is concluded that Cu partly substitutes Mg in the precipitates. Cu additions showed greatly enhanced formation of elongated precipitates for NA (one week) + BH. The hardening phases for the high-Cu alloy corresponds broadly to Q' precursors, the hardening precipitates of Cu-free alloy to β''. For short time AA (5 min 180 °C) Cu additions lead to increased number densities of clusters. The AA response after NA is concluded to be dominated by residual solutes in the matrix, for a Mg-rich alloy, without Cu additions, the lower Si residual is suggested to cause the lower AA response. [133] Recently, Jia et al. [132] reported for Cu added alloys a low influence of the Mg/Si ratio on the negative effect of NA on the AA response. [132] We conclude: Direct BH results in a high density of precipitates with a narrow Mg/Si ratio. In the T6 state a uniform distribution of needle like precipitates is observed. Prior NA causes a high number of small clusters with large Mg/Si-ratio spread after BH. For T6 a bimodal size distribution of few large elongated needle precipitates and short needle like precipitates is found. The impact of NA to further AA on APT results stagnates for NA $\geq$ 3 h. Cu incorporates into precipitates. Cu reduces the influence of the Mg/Si ratio on the NA + AA Summary and conclusion The occurrence of clusters can have important technological influence on the material properties of aluminum alloys. Ranging from natural aging in general, the rapid hardening reaction at artificial aging for AlCuMg alloys, over an increase in artificial aging response with prior natural aging for low alloyed AlMgSi alloys, to strongly decreased artificial hardening kinetics of high alloyed AlMgSi due to prior natural aging. In general the evolution of clusters at RT is well characterized by indirect methods as resistivity measurements, differential scanning calorimetry, positron annihilation lifetime spectroscopy, hardness and tensile testing. Hardness measurements are seen to be an easy way to follow the strength evolution of the material due to clustering. The strength to cluster relationship is often interpreted by means of a monotonous function, i.e. increasing strength contribution corresponds to more clusters. The increase of hardness is either proportional to the square root of the volume fraction of clusters (shearable obstacles), or direct proportional to the volume fraction of clusters (short range ordering contribution). Further, effects of clusters are seen from tensile testing, such as differences in the Portevin Le-Chatelier effect for as-

quenched and naturally aged alloys. Resistivity measurements can be used to study the evolution of clusters at the early stages due to high sensitivity and time-resolution. Resistivity is sensitive to cluster number density / inter-cluster distance. Although, the resistivity / hardness relationship for natural aging is seen to be alloy dependent. Hence, the drawback of the method is the weak connection between signal, cluster properties and mechanical properties. Positron annihilation spectroscopy is especially a sensitive tool in vacancy related processes, which play an important role for clustering. The positron lifetime signal over natural aging time can be interpreted as an overlaying decrease due to vacancy annihilation and an increase due to cluster formation. The increase in lifetime signal can be correlated to the hardness increase and thus the impact of additional alloying elements or substituting alloying elements can be studied. The effect of clusters on the following precipitation sequence is best revealed via differential scanning calorimetry. The peak corresponding to β'' is shifted to higher temperatures and endothermic traces in this region arise and increase with natural aging time. Also storage periods of short time pre-aged material for longer times at RT arise and increase endothermic traces. The most important technique for direct visualizing and measuring of clusters with near atomic resolution and gaining precise information of the compositions of clusters is atom probe tomography. Clustering in AlMgSi alloys is studied by numerous authors in detail. It is seen that effects caused by clusters, formed during RT, undergo a transient change during storage. Therefore, the evolution of clusters during RT is suggested to be grouped into distinct stages. A practical viewpoint is to treat the clusters formed at room temperature as their own "metastable state". Strong evidence exists that natural aging clusters revert upon elevated artificial aging temperatures. Their effect on artificial aging in AlMgSi alloys is attributed to solutes depletion, a concurrent cluster solution and precipitate forming reaction, and their interaction with vacancies. Most authors find that cluster formed during natural aging are difficult to transform to the major hardening phase upon artificial aging, while clusters formed during pre-aging at moderate temperature can transform to subsequent phases at artificial aging temperatures. From atom probe tomography results natural aging clusters are seen to stagnate in size early and increase further in number density in long term RT storage. Pre-aging clusters are seen to grow significantly with increasing pre-aging time. Often a narrow distribution (approx. unity) in the Mg/Si ratio is observed for pre-aging clusters, whereas for natural aging clusters a large Mg/Si ratio spread over the population is obtained. This is suggested to be the origin of non-transformable clusters, since later precipitates in AlMgSi alloys show often a ratio around unity. Other authors blame specifically the Si-rich clusters. Moreover, it is stated that the matrix Mg/Si-ratio influences the stability of NA clusters at artificial aging temperatures. For early cluster forming upon natural aging, Si is often suggested to have a leading role. Separate Si (Si-rich) and Mg (or at least Mg-rich) clusters are suggested to form, followed by transforming into Mg, Si co-clusters. Solute additions can influence the clustering (e.g. Cu is suggested to aggregate to clusters and to change the cluster chemistry

and their transformation upon artificial aging; Sn retards the formation of clusters due to vacancy trapping). With advanced analysis algorithms and improved methods new in use today, i.e. the latest developments in atom probe tomography detectors and cryo-atom-transfer probe equipment, further and more detailed insights into the nature of clusters will be possible in the near future.

Chapter 4

Experimental approach and applied methods

To access the time-range of natural aging below 100 min, it is necessary to produce the specimen under cooled conditions, and further use a cryo transfer system to the APT analysis chamber. Sample production maintaining a cold-chain is desribed in section 4.1. The approach for data analysis is described in section 4.2. Several data analyzing methods were applied and developed on a python script basis, as an independent analyzing tool. Main focus of the analyzing methods is to assess the spatial distribution of solutes in the sample and make a statement about the amount of clustered solutes.

4.1 Sample production & Atom probe tomography

Starting point for sample production is a EN-AW 6016 sheet material with a thickness of 1.25 mm provided by AMAG rolling GmbH. As a first step a so-called "blank" is produced from the sheet material.
The usual sequence: "blank" production → first step electro-polishing → second step electro-polishing; for APT sample production is used [95, 98]. Depending on the applied solution heat treatment strategy the sequence is modified. Starting from a blank, with the bulk aging strategy the sequence: blank → solution heat treatment → first step electro-polishing → first step electro-polishing; is used. Otherwise, if a nano scaled sample aging strategy is applied the sequence: blank → first step electro-polishing → second step electro-polishing → solution heat treatment → second step electro-polishing; is applied.

4.1.1 "Blank" production

Rough cutting of rectangular to quadratic pieces (to fit into the cutting machine) is done via a sheet shear or hand saw, maintaining the edge orientiations of the prior rolling product. Fine

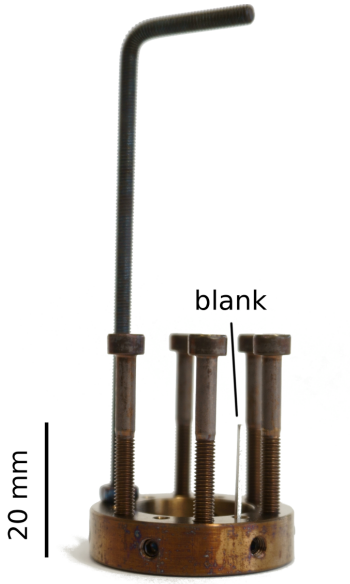

Figure 4.1: Solution heat treatment holder with a blank. Vertical srews are used as place holders, worm srews are used to mount the blanks in the holes.

cutting of pieces 20x20 mm is done via a Struers secotom cutting machine. The quadratic pieces are glued with a double-sided adhesive tape to a brass stamp, with a metric threat, and two nuts, made from tooling steel, which are used to adjust the height taken-off from the sheet thickness and ensure a preferably planar surface. The pieces are in this way grinded from both sides to the final dimension of 1 or 0.7 mm. Stripes with the same width as thickness are cut from the thinned pieces, with a horizontally adjustable mounting (smallest stepsize 0.05 mm), in the cutting machine. The stripes are, if needed, grinded manually, with or without the use of tweezers, to a final dimension difference of width to thickness of approximately $\pm 5\%$ in the middle of the blank, e.g. below a difference of 0.05 mm for a 1x1x20 mm blank. The blank is deburred along the length axes, and finally cleaned with iso-propanol in an ultrasonic bath. In Figure 4.1 a blank is shown in the holder for solution heat treatment. The produced blank is electro-polished to the final APT sample within a two-step method, except for samples reported in chapter 5. There another sample production strategy via focused ion beam (FIB) cutting was applied, but was concluded to be un-economical due to low yield (high fraction of fractured samples and if not fractured low measured amount of data).
The following sequence describes the production sequence used for maintaining a cold chain.

4.1.2 First-step electro-polishing

Initially intended to produce the final sample shape already within one electro-polishing setup, which is cooled, the developed first-step electro-polishing setup consists of the follwing

parts:

- sample holder (Figure 4.2 a))
- beaker with a Pt wire loop attached (Figure 4.2 b))
- isolated pot for LN_2 quenching (Figure 4.2 c))
- laboratory power supply (1.5-15 V, max. 1.5 A) (Figure 4.2 d))
- LN_2 transport dewar (Figure 4.2 e))
- power supply for a stepper motor
- microcontroller (Arduino Uno) and H-bridge (L293D)
- breadboard, with prototype electronic circuit
- connection cables
- power supply for the microcontroller
- double-wall vessel
- simple linear motion generator (Figure 4.3)
- stepper motor (SC2018S0604-A) with driving gear wheel (Figure 4.3 e))

The used circuit, for operating the electro-polishing setup, is shown in Figure 4.4. The used circuit for measuring the current during electro-polishing is shown in Figure 4.5. Without measurement of the current, the electrolyis (load) circuit is disconnected from the microcontroller and manually stopped, by turning-off the laboratory power supply. The used Arduino sketch is shown in section 8.13, based on Ref. [153–155]. The microcontroller is utilized to switch the H-bridge (L293D), which controls the current circuit with motor power supply and stepper motor. The other current circuit is used for sample production (load / electrolysis circuit). The circuit is switched by the microcontroller with a solid state relais (SSR) and the current is measured via voltage drop of a shunt resistor. The blank is connected as anode in a two-layer electrolysis, the bottom layer is electrical isolating (GALDEN HT-80) and the second layer, above the first, is conducting (25 % HNO_3 in methanol). The second electrolyte layer contains the cathode (Pt wire loop). The blank is submerged through the second layer and Pt wire loop into the first layer, see Figure 4.7. A few millimeters of the blank are in contact with the first layer. Only the region of the blank which is in contact with the conducting electrolyte is electrochemically dissolved in the electrolyte. At the cathode gas bubbles are created due to electrochemical decomposition of the electrolyte.

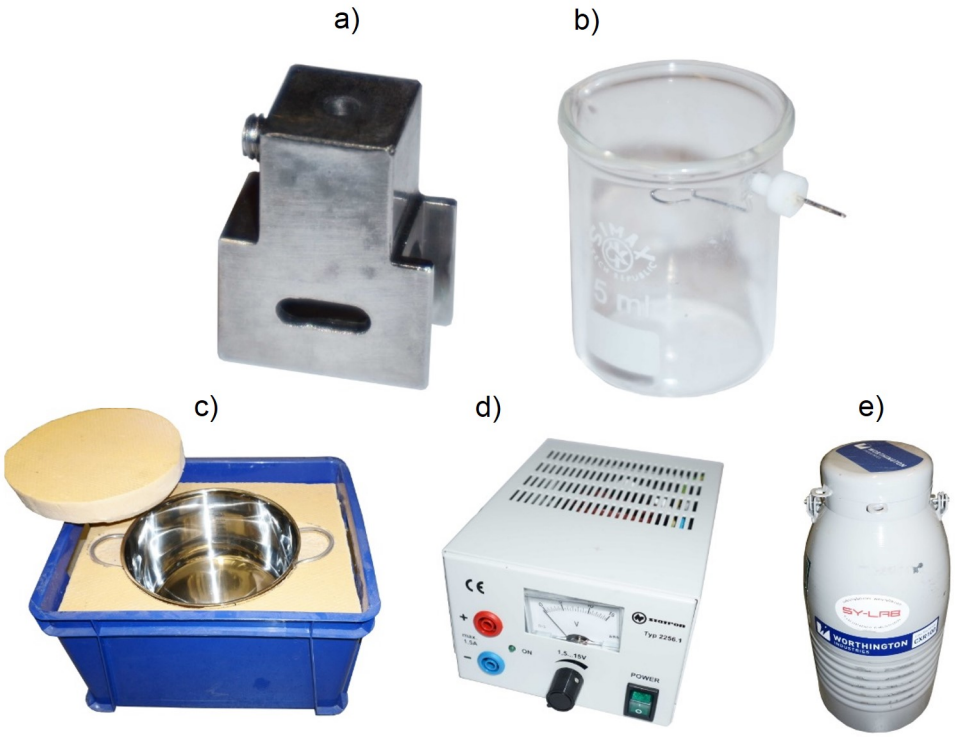

Figure 4.2: a) sample holder, b) beaker with Pt wire loop, c) isolated pot for LN_2 quenching, d) laboratory power supply, e) LN_2 transport dewar.

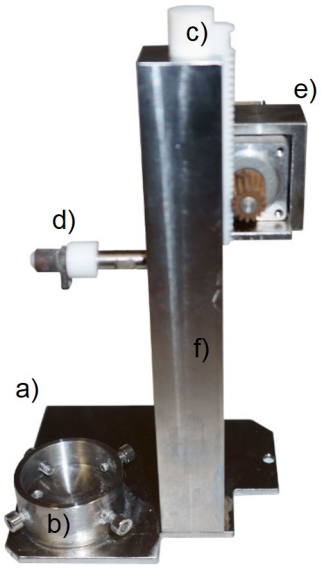

Figure 4.3: Linear motion generator: a) groundplate, b) beaker mounting, c) cylinder with driven gear rack, d) horizontal rod as mounting for the sample holder, e) stepper motor with driving gear wheel, and f) bushing.

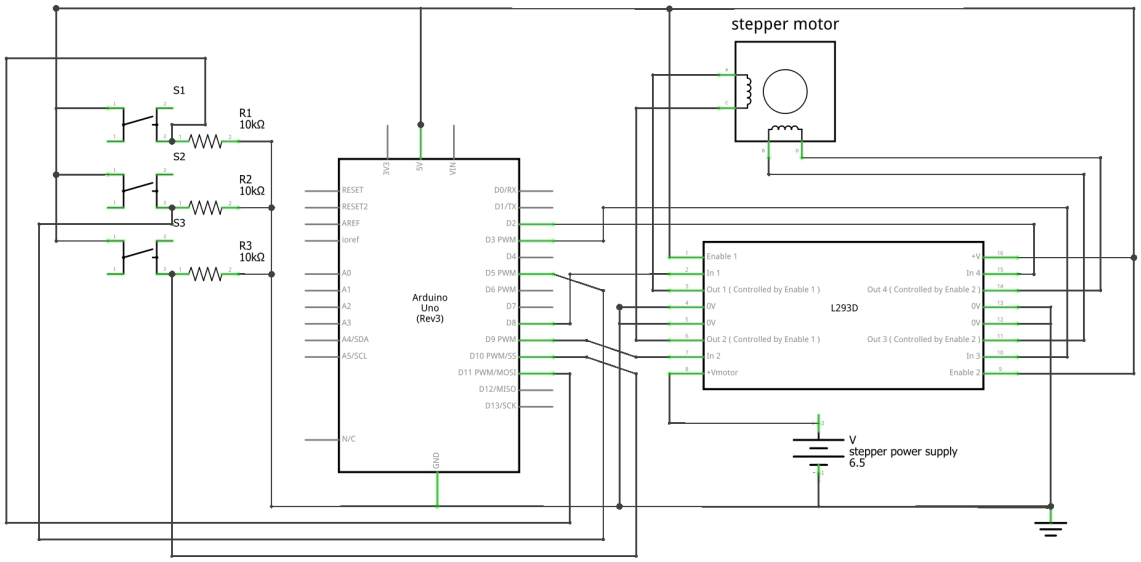

Figure 4.4: Circuit scheme (drawn with [156]) for stepper motor control. The microcontroller is utilized to switch the H-bridge (L293D), which controls the current circuit with motor power supply and stepper motor. The three switches are used as input for the movement and direction of the stepper motor (section 8.13, S1 up and S3 down).

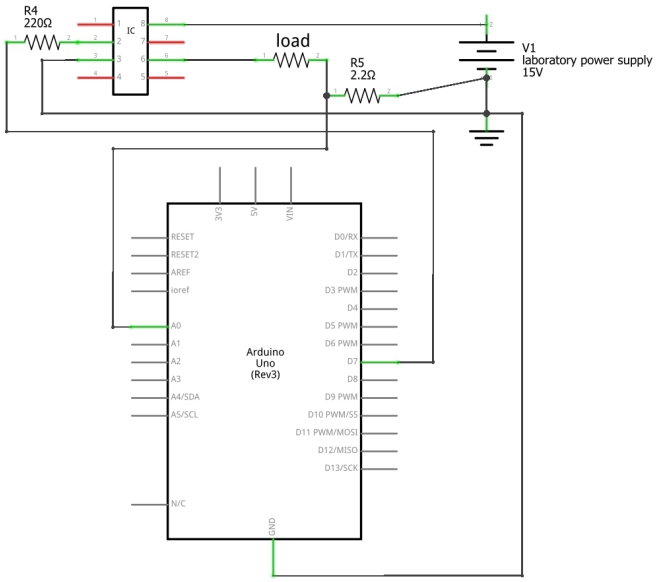

Figure 4.5: Circuit scheme (drawn with [156]) for current measurement and switching the electrolyis curcuit (load), which is powered by the laboratory power supply (V1). The circuit is controlled by the microcontroller with a solid state relais (IC) and the current is measured via voltage drop of a shunt resistor.

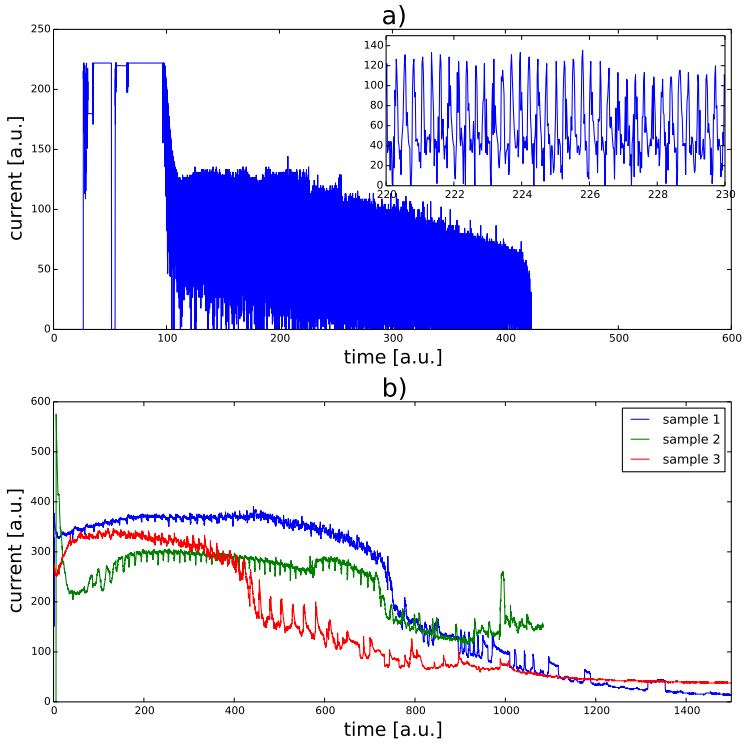

Figure 4.6: Measured current curves: a) electro-polishing with up-/down-movement which can be obtained in the current curve (insert) , b) electro-polishing without movement of the sample, additionally a software low pass filter is applied to the signal [153].

Sometimes also Cu is deposited on the Pt wire (dissolved Cu from the sample alloy), but if the eletrolysis is stopped, the Cu is resolved in the eletrolyte due to chemical etching of the acid. Additional gas production is obtained with higher currents. Higher currents heat the eletrolyte, due to faster solving of the sample and higher resistance heating, which leads to accelerated evaporation of the two electrolytes. Larger gas bubbles are obtained from the isolating layer, which start to expand and collapse at the blank surface, inducing perturbance to the electrolyte surface.

For the first electro-polishing experiments, which were conducted at room temperature, the blank was periodically moved down- and up-wards, to form a larger lateral thinned region. Solving of the material is not uniform in the conducting electrolyte, regions of increased removal are electrolyte/electrolyte interface, electrolyte/air interface and regions where the distance from cathode to anode is small. This movement approach was neglected in the further experiments, to investigate solely the current time curve from the electrolysis, otherwise the signal is overlaid with the change of metal/electrolyte interface area, due to the up and down movement (see Figure 4.6 a)). A stopping criteria for the electrolysis circuit would have been needed to use the first-step electro-polishing setup as the sole preparation setup at low temperatures, but from the gained current-time curves it could not be determined

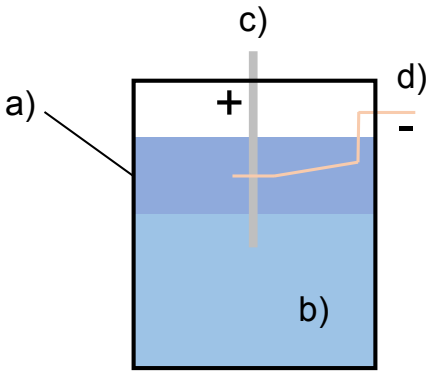

Figure 4.7: First-step electro-polishing: a) electrolyte, b) inert layer, c) blank, d) Pt wire loop.

when the blank intersected (see Figure 4.6 b)) and no criteria could be deduced. Problems are the low resolution of the measurement system, gas bubbles, which cause spikes in the current-time curve, and evaporation of too much electrolyte during electro-polishing of "thick" 1x1x20 blanks. Additionally, it could not be ensured that the produced samples are sharp enough for the use as APT tips. Therefore the first-step electro-polishing is only used as pre-step for final sample production and the solely use of the first-step setup for final sample production would need further development. After first-step electro-polishing the blank is cleaned either via rinsing topside down with iso-propanol or submerging the tip with tweezers into iso-propanol in an ultrasonic bath.

4.1.3 Second-step electro-polishing

Due to the unknown needed development time to solve the stopping criteria and tip size issue, it was decided to follow another heat treatment strategy (nano aging) as already mentioned: Blanks are first-step electro-polished (Figure 4.7) with manual stopping of the eletrolysis. A neck is electro-polished in a second step (micro-polishing with 2% $HClO_4$ in 2-butoxyethanol), with a diameter in the order of 5 to 20 μm (Figure 4.10) near the apex and then the sample is solution treated and quenched.
As "second step electro-polishing", horizontal electro-polishing under a modified optical light microscope (micro-polishing) is done in a cooling chamber at -40 °C. The microscope was modified by adding a mounting cylinder for the sample holder (same sample holder as for first-step electro-polishing) at the xy-table and a stationary Pt wire loop, so that a relative movement of the sample is possible, see Figure 4.8. For the use at -40 °C the xy-table was dissassembled, the lubrication grease removed and re-assembled again. The height of the Pt loop is chosen so that the ideal sample position is at the height of the center of the loop,

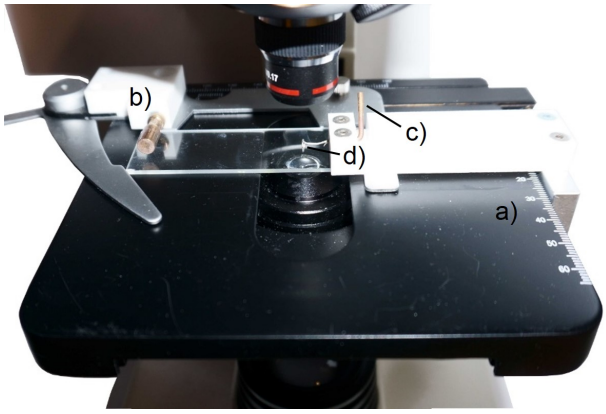

Figure 4.8: Micro-polishing setup: a) xy-table, b) mounting for sample holder (free to move, anode), c) connection to Pt wire loop d) (static, cathode).

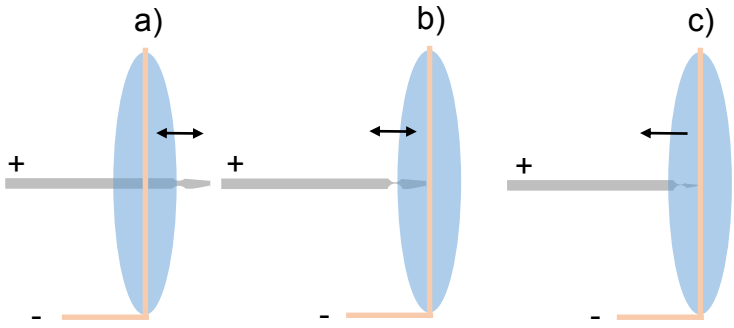

Figure 4.9: Micro-polishing procedure: a) a neck is formed on the blank, b) the neck is further thinned, c) pulsing and final separation.

due the fact that every blank is not exactly at the ideal position in the sample holder hole, the z position of the sample tip is changed with slight tilting of the sample holder on the mounting cylinder. The mounting cylinder and the Pt wire loop mounting can be connected via crocodile clip cables to the power supply. At the anode connecting cable a self resetting push-button is installed.

A droplet of electrolyte is pippeted into the wire loop and hold in the loop due to the surface tension of the electrolyte. For micro-polishing at -40 °C 3% $HClO_4$ (72%), 16% 2-Ethoxyethanol, 22% 1,2 Dimethoxyethan in methanol is used as electrolyte (at room temperature 2% $HClO_4$ in 2-butoxyethanol). 2% $HClO_4$ in 2-butoxyethanol was also tested at low temperature, but showed too high viscosity, 25 % HNO_3 in methanol was shown to be too highly concentrated for the use as second step electrolyte. The applied voltage ranges from 5 to 7 V. The blank is moved through the elecrolyte so that a bit of the blank is outside on the other side of the droplet (Figure 4.9 a)). A neck is formed through slight back and forward movement of the blank with a xy-table, while the push-button is pressed to close the electric circuit. If the neck is sufficiently shaped and thinned, the blank is moved backward

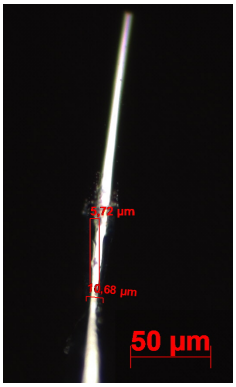

Figure 4.10: Optical light microscope picture of the sample after micro-polishing a neck into the sample. The necked region is used as starting point for final APT sample production in a cooling chamber.

so that the apex and neck is in the elecrolyte, further the neck is thinned by continous closing of the circuit and back and forward movement (Figure 4.9 b)), the process is interrupted to optical controll the thickness of the necked region by partly moving out of the electrolyte. When the neck is almost invisble, or invisble in the optical microscope, the blank is moved back into the electrolyte and only single impulses are used, accompanied by moving out of the electrolyte (Figure 4.9 c)), the APT sample is finished when the front part of the necked region is in this way separated. The finished tip is finally cleaned by topside-down rinsing with iso-propanol and is put back into the sample holder and the liquid nitrogen transport dewar. The electrolyte droplet is removed with the angle of laboratory paper tissue.

4.1.4 Solution heat treatment and quenching

Solution heat treatment of blanks or various types of APT-tips is carried out in an air furnace with N_2 purging at 545 °C for 15 min. The air-furnace is pre-heated for two to three hours and the N_2 purging started 30 min before the first solution heat treatment cycle. The sample holder is put onto a steel sheet in the air furnace, the furnace closed, waited for 15 min, the sample holder taken from the furnace and plunged into LN_2 (in an isolated pot, Figure 4.2 c)). The isolated pot is positioned elevated under the furnace to reduce movement time at room temperature air. The sample holder is put, under LN_2 in the pot, into a one-side open cylindrical plastic container with an attached wire. The plastic container is put into the pre-filled LN_2 transport dewar (Figure 4.2 e)).

4.1.5 Artefacts of sample production

Sometimes artefacts of sample production cannot be prevented completely. Some artefacts from sample production can make the measurement of the sample impossible, others create artefacts in the gained dataset which can be addressed by data analysis.

The measurement is impossible if the sample is bent or the tip is not sharp enough. Several reasons for bent samples are imaginable. Most of them conserne handling issues, some of them are listed in the following:

- accidential mechanical touch of the tip (holder or tweezers)
- tipping the finished specimen top-side forward into cooled iso-propanol
- re-filling the transport dewar with liquid nitrogen from above
- if the front part of the necked region is pulled towards the surface of the droplet during the final pulsing during micro-polishing

Too blunt sampled are sometimes obtained when the sample is thick, or only a small thinned region exists after first-step electro-polishing. Sometimes a semi-transparent film forms at the tip, which generally hinders electro-polishing, or shades the electro-polishing of the metal and leads to blunted APT tips.
A typical artefact from electro-polishing, often obtained, is a Cu cap at the apex of the APT tip. Cu is present in the alloy EN-AW 6016 and gets dissolved during electro-polishing and is accumulated in the elecrolyte. Additionally to the electrochemical dissolution of Al into the elecrolyte ($Al \rightarrow Al^{3+} + 3e^-$), Cu in solution near the apex, Cu^{2+} or Cu^+, can oxidize Al via cementation: $3\ Cu^{2+} + 2\ Al \rightarrow 3\ Cu + 2\ Al^{3+}$. An extrem example for a Cu cap is seen in Figure 4.11.

4.1.6 APT experimental parameters

Most experiments are run in voltage mode with a pulse fraction of 20 %, 200 kHz and a detection rate of 1% at a temperature of 30 K. The sample is approximately positioned to the typical sample position centered in front the local electrode, further the run is started and fine positioned. The tip is aligned so that the hitmap slightly underfills the detector space concentric.
Laser measurements were tested (250 kHz, 30 K, 532 nm wavelength and 1 nJ Laser energy based on conclusions from [157]), but lead to unsatisfying results concerning chemical position (see Table 4.1) and additionally, increased Si migration effects were suspected [113]. The difference in composition for the laser measurements are speculated to be due to different charge state distributions. Increasing the amount of monovalent charged ions, leading to the loss of the major Si^+ ions in the AlH^+, AlH_2^+ peaks and a shift from detected $^{24}Mg^2+$ to $^{24}Mg^+$.

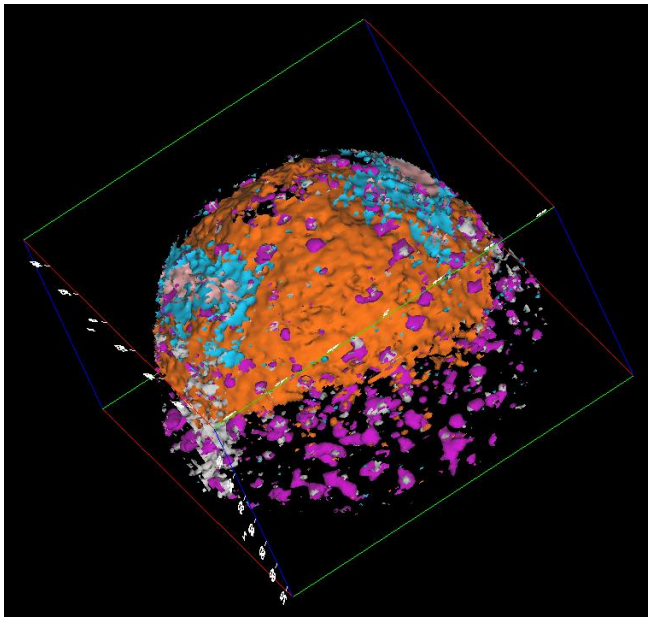

Figure 4.11: Cu cap and oxide shell at the tip apex of a pre-aged material, obtained with laser measurement. Iso-surface colors: Cu in orange, O in cyan and H_3 rose, Mg in magenta.

Table 4.1: Compositions generated by two laser measurements compared to a voltage measurement of the same material. Compositions measured for different heat treatment states for the same material can be obtained in Table 6.1.

comp. [at.%]	Laser 1 (R21_09112)	Laser 2 (R21_09114)	voltage (R21_09042)
Mg	0.37	0.39	0.34
Si	0.60	0.49	0.88
$^{24}Mg^{+}/^{24}Mg^{2+}$	0.165	0.177	0.016

4.2 APT data analysis

Basis for data analysis is the reconstruction in form of the position file of the atoms in the sample. As a first step, after a successful measurement, for data analysis the gained .RHIT file needs to be processed to a reconstruction (.pos file, .epos file). This is done with the commercial software IVAS^{TM} (3.6.12) and desribed in the following section 4.2.1.
Starting from the generated .pos or .epos file further customized data analysis is applied (section 4.2.1).

4.2.1 From .RHIT to .pos and .epos

The steps from .RHIT to .pos file are already well described in [96, 158], but hence every change in one of the steps can possibly change the pos file, the usually applied options within this thesis are shortly described in the following sections with the purpose of each step.

4.2.1.1 Selection of a ion sequence range

At the beginning of an experiment the aluminum alloy APT tip is not in a field equilibrated shape, additionally often a oxide layer and/or a Cu cap is present. With increasing experiment time the tip shape equilibrates itself. Therefore the start of the experiment, the first few 100000^{th} counts, are often neglected. [96]
In the background signal there is often a significant drop after a given number of field-evaporated atoms visible, this drop is often near the suggested starting sequence number. If a Cu cap is present, it can often be cut away by simply choosing a higher sequence starting number. In some cases the end sequence number needs to be adjusted if a tip fracture was not detected or a hot-spot on the detector hit map occured. Another exception occures, if a part of the tip breaks off, but the tip recovers. Best approach is to generate two reconstructions, before and after the fracture. If the two parts are reconstructed at once in one .pos file, unreal neighborhood relations are created. Although, the latter approach should not make a large difference for small feature sizes as for clustering.

4.2.1.2 Selection of a detector region of interest

From the detector hitmap an automatic fit of an ellipse, as chosen detector space, is suggested by IVAS^{TM}, hits outside this ellipse are neglected. The detector space area is calculated from this ellipse, which is later used in the reconstruction process.
The automatic fit is accepted if the sample is sufficiently circular. [96, 158]

4.2.1.3 Time-of-flight to mass-to-charge ratio (m/n)

Different voltages at departure of the ion lead to different start accelerations, therefore the time-of-flight (tof) spectrum is calibrated for voltage. The spectrum is calibrated by the flight-time of a known calibration material (Al). Due to the changed flight paths for a 2D detector, compared to a linear one-dimensional case, also a flight path calibration, depending on the position of the hit on the 2D detector, is done (bowl correction).
The standard time-of-flight conversion is usually leading to sharp enough peaks (mass resolution for full-width at half maximum (FWHM) >1000) and accepted. [96, 158]

4.2.1.4 Correction of the mass-to-charge ratio

In this step a piecewise linear interpolation function is applied to the gained mass-to-charge ratio histogram, shifting the known peaks to the exact positions. All the known elemental (m/n) peaks are chosen and mapped to their ion species. The spectrum in-between smallest and largest m/n is fitted piecewise with the chosen peaks as nodes. Only known elemental peaks should be assigned at this step, in general only low changes of the histogram on the m/n axis are expected ($<\sim$ 0.2 Da).

Table 4.2: Assigned ion species to peaks for m/n calibration.

ion species	approximate m/n [Da]
$^{1}H^{+}$	1
$^{2}H^{+}$	2
$^{24}Mg^{2+}$	12
$^{25}Mg^{2+}$	12.5
$^{26}Mg^{2+}$	13
$^{27}Al^{2+}$	13.5
$^{28}Si^{2+}$	14
$^{29}Si^{2+}$	14.5
$^{30}Si^{2+}$	15
$^{27}Al^{+}$	27
$^{63}Cu^{+}$	63
$^{65}Cu^{+}$	65

The ion species as shown in Table 4.2 are assigned at this stage for correction of the mass-to-charge ratio. [96, 158]

4.2.1.5 Ranging

Usually in atom probe tomography a known material, in terms of chemical compositions, is investigated. During the ranging step a map from mass-to-charge ratio to the ion species is constructed, $m/n \rightarrow$ [element/complex]$^{n+}$. Often the natural abundance of elements is used to identify considered peaks. The problem of peak identification for complex ion peaks is strongly related to the *Knapsack problem* [159]. For atom probe tomography the combinations of a given set of possible elements is mapped to the unknown given m/n of a peak via application of a charge number (e.g. $^{+}$,$^{2+}$ ...) for possible solutions, here especially the uncertainty of the m/n histogram needs to be considered [159].
In some special cases methods use additional information, besides the mass-to-charge ratio, for the mapping. For example $(x, y, z, m/n) \rightarrow$ [element/complex]$^{n+}$ [160] or multiple hit information $\rightarrow$ [element/complex]$^{n+}$ [161] is used.
Additionally, a special side-effect of atom probe measurements is the occurence of H in every sample measured. This is due to the presence of H in the UHV analysis chamber. Hydrogen out-diffuses from the steel of the chamber.
The range file used by default is shown below.

```
[Ions]
Number=8
```

```
Ion1=Al
Ion2=H
Ion3=Mg
Ion4=Si
Ion5=Cu
Ion6=V
Ion7=Ga
Ion8=O
Ion9=Mn
[Ranges]
Number=24
Range1=13.4500 13.6000 Vol:0.01661 Al:1 Color:33FFFF
Range2=26.9210 27.0990 Vol:0.01661 Al:1 Color:33FFFF
Range3=0.9850 1.0950 Vol:0.00000 H:1 Color:CCCC00
Range4=1.9790 2.1090 Vol:0.00000 H:2 Color:FF0000
Range5=11.9550 12.0500 Vol:0.02325 Mg:1 Color:CC00CC
Range6=12.4560 12.5750 Vol:0.02325 Mg:1 Color:CC00CC
Range7=12.9430 13.1070 Vol:0.02325 Mg:1 Color:CC00CC
Range8=23.8970 24.0490 Vol:0.02325 Mg:1 Color:CC00CC
Range9=25.9310 26.0350 Vol:0.02325 Mg:1 Color:CC00CC
Range10=13.9530 14.0790 Vol:0.02003 Si:1 Color:CCCCCC
Range11=14.4490 14.5490 Vol:0.02003 Si:1 Color:CCCCCC
Range12=14.9530 15.0440 Vol:0.02003 Si:1 Color:CCCCCC
Range13=62.8120 63.1100 Vol:0.01181 Cu:1 Color:FF6600
Range14=64.8100 65.1500 Vol:0.01181 Cu:1 Color:FF6600
Range15=25.4170 25.5330 Vol:0.01382 V:1 Color:CC00CC
Range16=68.7850 69.0580 Vol:0.01960 Ga:1 Color:FFFF00
Range17=70.8660 70.9820 Vol:0.01960 Ga:1 Color:FFFF00
Range18=27.9400 28.1760 Vol:0.01661 Al:1 H:1 Color:00FF00
Range19=28.9280 29.2410 Vol:0.01661 Al:1 H:2 Color:0000FF
Range20=17.9490 18.3500 Vol:0.02883 H:2 O:1 Color:CCCC00
Range21=24.9310 25.0350 Vol:0.01767 Mg:1 Color:CC00CC
Range22=27.4900 27.6500 Vol:0.01201 Mn:1 Color:CCCC00
Range23=16.9000 17.200 Vol:0.01382 V:1 Color:CC00CC
Range24=43.9000 44.2000 Vol:0.05636 Al:1 O:1 H:1 Color:35A9BD
```

This range file corresponds to the: $^{1}H^{+}$, $^{2}H^{+}$, $^{24}Mg^{2+}$, $^{25}Mg^{2+}$, $^{26}Mg^{2+}$, $^{24}Mg^{+}$, $^{25}Mg^{+}$, $^{26}Mg^{+}$, $^{27}Al^{2+}$, $^{27}Al^{+}$, $^{28}Si^{2+}$, $^{29}Si^{2+}$, $^{30}Si^{2+}$, $^{63}Cu^{+}$, $^{65}Cu^{+}$, $^{69}Ga^{+}$, $^{71}Ga^{+}$, $^{51}V^{2+}$, $^{51}V^{3+}$, $^{55}Mn^{2+}$ species, and further to the complex ions: AlH^{+}, AlH_2^{+}, $AlOH_2^{+}$, H_2O^{+}.

The $^{24}Mg^{+}$, $^{25}Mg^{+}$, $^{26}Mg^{+}$, $^{51}V^{2+}$, $^{51}V^{3+}$, peaks are often close to the background limit and the identity is not fully verified and also possible overlaps of Ti and V and Mg are possible. Possible overlays from Si^{+} with $^{27}Al^{+}$, AlH^{+}, AlH_2^{+} cannot be accounted for, but no peak is obtained at 30 Da, which implicates that there is only very low amount of Si^{+} possible. This is also true for eventual Fe^{2+} overlays. Here no significant peak at 28.5 Da is seen, which also implicates that there is only very low or no Fe in the matrix. Often no peak is seen at 27.5 Da, because the signal is lost in the tail of the $^{27}Al^{+}$ peak. If the peak is obtained, it is ranged as $^{55}Mn^{2+}$, but sometimes a slight left-shift of the ideal position is seen. When a large Cu cap is obtained, often also the Cu^{2+} and CuH^{+} peaks can be obtained. The $^{69}Ga^{+}$, $^{71}Ga^{+}$ peaks are mainly obtained if the sample is produced by focused ion beam, but also present in technical alloys as residual from primary Al production. $AlOH^{+}$, H_2O^{+} appear and dissapear from measurement to measurement.
For spatial analysis the $^{27}Al^{+}$, $^{27}Al^{2+}$, $^{28}Si^{2+}$, $^{29}Si^{2+}$, $^{30}Si^{2+}$, $^{63}Cu^{+}$, $^{65}Cu^{+}$, $^{24}Mg^{2+}$, $^{25}Mg^{2+}$, $^{26}Mg^{2+}$ peaks are of major interest, therefore the peaks are corrected to their exact positions (as described in the previous step) and always the same ranges for them are used. Still the "unsure" Mg^{+} peaks are used, but their relative amount in comparison to the whole identified Mg is irrelevant.

4.2.1.6 Reconstruction

The reconstruction protocol builds the 3D coordinates from the voltage at evaporation, 2D detector coordinates, the ranged species and the sequence of evaporation (X, Y, V,ranging, sequence $\rightarrow x, y, z$). In this thesis the reconstruction protocol via voltage curve is applied. [95, 111, 162]
The current radius of the specimen is deduced from the field evaporation value F_e, field factor k_f and voltage V. For the i^{th} atom the equation 4.1 is valid. A similar approach is to use a voltage curve, fitted to the experimental curve, to ensure monotony in the evolution of R.

$$R_i = \frac{V_i}{F_e k_f} \tag{4.1}$$

From the detector coordinates X_i, Y_i the distance from the detector origin D_i is calculated, Equation 4.2.

$$D_i = \sqrt{X_i^2 + Y_i^2} \tag{4.2}$$

The compressed angle is calculated via θ_i' as in Equation 4.3, with L as flight path length and ξ as image compression factor (ICF). (Figure 4.12)

$$\theta_i' = \arctan\left(\frac{D_i}{L + \xi R_i}\right) \tag{4.3}$$

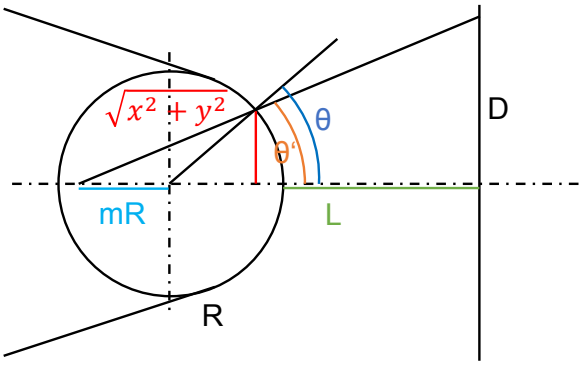

Figure 4.12: Sketch of the applied projection law. [95, 162]

With θ_i (Equation 4.4) , R_i and D_i the x and y coordinates in the sample of the ion can be computed (Equation 4.5 and Equation 4.6).

$$\theta_i = \theta_i' + \arcsin\left((\xi - 1)\sin\theta_i'\right) \tag{4.4}$$

$$x_i = \frac{X_i}{D_i} R_i \sin\theta_i \tag{4.5}$$

$$y_i = \frac{Y_i}{D_i} R_i \sin\theta_i \tag{4.6}$$

The z_i coordinates are computed with Equation 4.7 with the use of Equation 4.8 and 4.9, Ω_j is the specific atomic volume of the species, S_D the detector area and η the detection efficiency.

$$z_i = \sum_{j=0}^{i} (dz_j) + \Delta z_i \tag{4.7}$$

$$dz_j = \frac{\Omega_j}{V_j^2} \frac{L k_f^2 F_e^2}{\eta S_D \xi^2} \tag{4.8}$$

$$\Delta z_i = R_i \left(1 - \sqrt{1 - \frac{x_i^2 + y_i^2}{R_i^2}}\right) \tag{4.9}$$

The above mentioned equations are only valid if the APT tip axis lies along with the detector space origin at X=0,Y=0.

As user input variables k_f and ξ needs to be chosen. Therefore the values are calibrated as described by [109]. For this purpose the "expert reconstruction explorer" within IVAS^{TM} is used.

An evaporation field fixed with 19 V/nm (for Al at 60 K) is used, this is possible because always the product $F_e k_f$ appears together in all equations and so effectively the product is calibrated [109]. The calibration process was developed for the straight flight path version of the local atom probe tomography (LEAP^{TM}), but within this thesis only the LEAP^{TM} versions equipped with a reflectron are used. Therefore the standard L input variable of 382 mm in the IVAS^{TM} is used for the reconstruction, but a virtual flight-path $L_{\text{virt.}}$ of ~44-50 mm (see section 4.2.2.2) for the calibration of the image compression factor ξ is utilized.

At least three poles $(h_i k_i l_i)$ are chosen and the observed angles between them are calculated [163] via Equations 4.10 and 4.11. This approach is only possible for crystal systems

where the vector [hkl] is perpendicular to the plane (hkl), this is full-filled for the cubic systems. Further a mean from theoretical/observed angle is calculated and used as image compression factor, Equations 4.12 and 4.13.

$$D_{ij} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}, i \neq j \tag{4.10}$$

$$\theta_{\text{obs.},ij} = \arctan\left(\frac{D_{ij}}{L_{\text{virt.}}}\right) \tag{4.11}$$

$$\theta_{\text{theoret.},ij} = \arccos\left(\frac{h_i h_j + k_i k_j + l_i l_j}{\sqrt{(h_i^2+k_i^2+l_i^2)(h_j^2+k_j^2+l_j^2)}}\right) \tag{4.12}$$

$$\xi \sim \text{Mean}\left(\frac{\theta_{\text{theoret.},ij}}{\theta_{\text{obs.},ij}}\right) \tag{4.13}$$

A large standard deviation Std $\left(\frac{\theta_{\text{theoret.},ij}}{\theta_{\text{obs.},ij}}\right)$ indicates a wrongly identified pole.
For pole identification a manual comparison of the gained detector hitmap to a known indicated detector hitmap, e.g. the hitmaps in Ref. [111], is done. In general only (hkl) for the conventional fcc crystal are valid for the calculation of the smallest interplane distance for the correlated direction, if h, k and l are all either even or odd. Usually a subset of the major poles {(002), (111), (022), (113)} is visible on the detector hitmap. For identification typical patterns are used: the (002) pole shows a typical 4-fold symmetry, (111) a 3-fold symmetry and the (022) pole is always accompanied by the two (133) minor poles in the direction of the (111) poles. Further (002), (022) (with the two minor (133) poles) and (111) usually build-up an easy identifiable triangle. Additionally, a Si surface migration artefact (Si is accumulated at this detector hitmap positions) is seen at the (111) pole and the (111)-(022) zone lines, which form a star-like shape. In cases of uncertainty, also a measurement of the volume (V) (MATLAB$^{\text{TM}}$ scripts discussed in section 4.2.2) of the final reconstruction for a correctly obtained volume in comparison to Equation 4.14 is reasonable.

$$V = \frac{\sum_i \Omega_i}{\eta} \tag{4.14}$$

The correct identification of the poles is the crucial part in the calibration of the reconstruction.
The so-gained ξ is used as input for the reconstruction. A start value for k_f is chosen, for the used alloy and temperature, a starting value of 5.0 is used. In the "expert reconstruction explorer" a subset of the reconstruction is chosen, based on two detector coordinates and an inner and outer radius of a circle. The inner radius is set to 0.0 mm and the outer radius to 2 mm, for the detector coordinates. The respective pole detector coordinates are used. Further the relative position of the center of the sub-sequence and the boundary length in percent can be set, here the standard values are used. The projection centers are also changed to the respective pole detector coordinates. Thus the z-coordinate of the reconstruction is build perpendicular to the lattice planes of the chosen pole and the inter-plane separation can be measured via the z spatial distribution map (z-SDM). The theoretical inter-plane

distance d_{hkl}, with the conventional lattice parameter a, is calculated with Equation 4.15.

$$d_{hkl} = \frac{a}{\sqrt{h^2 + k^2 + l^2}}, a = 0.405 \text{ nm} \tag{4.15}$$

It should be noted that the used a is actually the lattice parameter for room temperature, and is expected to be lower at 30 K. Also the APT tip, which is a single crystal at this scale, is under high tension due to the high applied field, leading to direction dependent elastic strains. But this considerations are neglectable, because the error made by the measurement of the interplane distance via z-SDM is larger than the further mentioned assumptions.
The inter-plane distance is measured as the x-distance from the peak at the origin (x-axis = 0) to the first peak. The measured inter-plane distance for the chosen poles are compared to the theoretical values and a corrected $k_{f,i+1}$ can be calculated based on the prior used value, see Equation 4.16. The mean from the different poles is calculated for the $k_{f,i+1}$ and as new starting value used. This procedure is repeated until the $d_{hkl,\text{obs.}}$ are acceptable accurate.
Also possible is simply adjusting the R_0 (initial radius of the reconstruction) scrollbar so that the first peak of the z-SDM for a pole is at the theoretical postition. The R_0 value is equivalent to a k_f value, which is automatically calculated when a k_f is set. The mean of the k_f gained in this way can be used for the final reconstruction (only small deviations are expected for different poles). When poles are far from the detector origin, the k_f of the pole nearest to the detector origin can be used: Usually a differing position from the pole position as reconstruction center in the final reconstruction (close to the detector center) is used, and the adjusted d_{hkl} in the "expert reconstruction explorer", will match better to the obtained in the final reconstruction.

$$k_{f,i+1} = \sqrt{k_{f,i}^2 \frac{d_{hkl,\text{obs.}}}{d_{hkl,\text{theoret.}}}} \tag{4.16}$$

With k_f and ξ the final reconstruction is built. From the .pos file an .epos file can be generated, which contains extra information from the .RHIT file, besides x, y, z, m/n, and can also be used by customized data analysis.

4.2.2 Customized data analysis

Data analysis methods based on the scripts of [164] were developed to be able to further refine existing methods, simplify applying the same data analysis for different pos files, to have a independent data analysis method besides the commercial IVAS$^{\text{TM}}$ and to create in-depth knowledge of the applied algorithms. The programming language python is used.
The script apt_importers.py (see section 8.2) is intended to contain static methods, which are imported by other scripts for data analysis. Original methods from [164], are the read_pos(), read_epos(), read_rrng(), label_ions(), deconvolve() in apt_importers.py and

volvis() from Vis.py (described in section 8.8). They depend on the pandas, numpy and VisPy python libraries.
The principle application sequence for a typical analysis is starting from ranging_kryo_proto.py (section 8.5) for checking the range file and possible additional peak identification. The next step is to create the detector hitmap plots and cutting, based on the detector hitmaps, with plot_multiple_hits_Si.py, section 8.3. For assessment of the spatial distribution of the solute atoms proto_function_RDF_data.py (section 8.7) is used, which creates radial distribution functions (RDF) in cumulative form and k-nearest neighbor distributions (kNN), for specified interactions, pos and range files.
For multiple ion analysis multiple_ion_analysis.py (section 8.6) can be used, but it was rarely applied in this work. In this thesis measurements are carried out in voltage mode and multi ion analysis has a more greater application potential for laser measurements. Also the current implementation is slow. The script analyse_recon.py (section 8.4 is used for determining the virtual flight path length $L_{\mathrm{virt.}}$. In the script largeSDM.py (section 8.9) analyses and tests for the application of a SDM-like method are done. Further with C14_art.py (section 8.10) precipitates can be analyzed, regarding ordering, and compared to an artificially generated C14 crystal structure. Feature identifiction (precipitates, cluster) is usually done with the MATLAB$^{\mathrm{TM}}$ [165] scripts clusteranalyse_AlMgSi.m (section 8.12), which are based on the cluster identification and analysis methods from Peter Felfer [124].

4.2.2.1 Spatial analysis

The heart of the spatial analysis methods is the python binding, pyflann, to the FLANN (Fast Library for Approximate Nearest Neighbors) [166], which allows to query the neighboring points, of a given point, in a point cloud (based on a kdtree [140]) (see e.g. proto_function_RDF_data.py section 8.7). An important detail is that the flann object is always created with the algorithm=4 parameter, which uses the exact kdtree and not a "approximate" variant. Also the flann object is always created with the euclidian metric. It should be noted that FLANN works with squared distances. Computing the root of the squared distances is unnecessary for identification of the k-nearest neighbor (monotony) and computationally expensive.

Radial distribution functions (RDF) and k-nearest neighbor (kNN) distributions

For the generation of RDFs and kNN distributions the proto_function_RDF_data.py (section 8.7)) script is used. It calls the methods calcRDF() and createNNHist() from apt_importers.py (section 8.2). The equations regarding the radial distribution function are described in section 6.6.2, a mathematical representation of the k-nearest neighbor

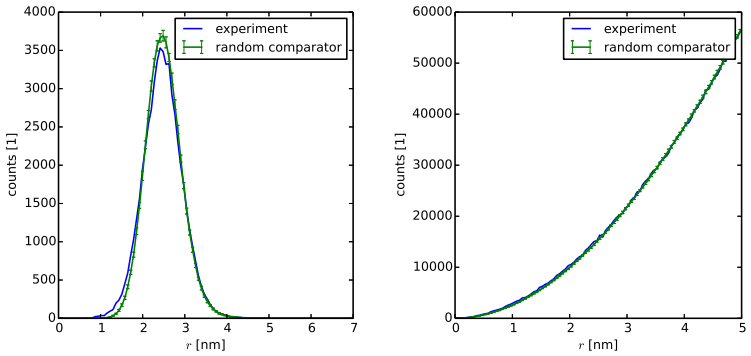

Figure 4.13: 5^{th} nearest neighbor distribution (k=5 NN) and radial distribution function (RDF) for a given reconstruction (Mg-Mg), with the respective random comparators.

distribution is shown in Equation 4.17.

$$\text{kNN} = \text{Hist}\left(\left|\left|\vec{P}_{\text{k-th neighbor}} - \vec{P}_i\right|\right|_2\right) \tag{4.17}$$

The generation of random comparators is executed via random labellig, i.e. choosing randomly from existing 3D positions, it should be noted that only ranged atoms are used for possible sites and the background positions are neglected in proto_function_RDF_data.py. For the estimation of the breadth of drawn possible random distributions, 40 random distributions are drawn. The mean curve of the curves is used as the "random distribution" and the standard deviation of the curves as boundaries for the breadth of possible random distributions. The results of the RDFs and kNNs are saved as .txt files. Figure 4.13 shows kNN and RDF distributions with the respective random comparators.
An interesting observation regarding the random distribution of the kNN is seen with IVAS$^{\text{TM}}$. When a .pos file analysis is opened and a nearest neighbor analysis is generated and saved, the program closed and the procedure repeated, then the two saved random curves of the kNN analyses match exactly. If the random comparator is built with random labeling, this would be rather an unlikely event, and would only occur if the random number generator is seeded with a constant.

SDM-like functions

SDM [107] -like functions, getRotateXYSDM() and getRotateZSDM() (apt_importers.py, section 8.2) are used by largeSDM.py (section 8.9) and C14_art.py (section 8.10). The two methods rely on the output of getDeltasSDMLarge() (apt_importers.py, section 8.2). The original approach of subdivision of the volume in [107] of the point cloud is not used. A point set and a maximum nearest neighbor number is passed to getDeltasSDMLarge(), and with FLANN, the interatomic difference vectors (Equation 4.18 and 4.19) up to the given maximum nearest neighbor are calculated for each point of the set (Equation 4.20), self-correlation (seeing itself) is neglected. This leads a $n_{\text{points}} N_{\text{max.NN}}$ long list of interatomic

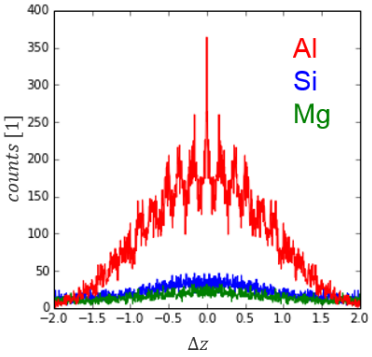

Figure 4.14: Obtained z-SDM from a selected precipitate for given ψ, θ.

difference vectors, re-written in Equation 4.21.

$$\vec{\mathrm{d}}(\vec{P}_i, \vec{P}_j) = \vec{P}_i - \vec{P}_j \tag{4.18}$$

$$\vec{\mathrm{d}}(\vec{P}_i, \vec{P}_j) = [x_i - x_j, y_i - y_j, z_i - z_j] \tag{4.19}$$

$$i = 1...n_{\text{points}}, j = n_{\text{1stNN}}...n_{N_{\text{max.NN}}} \tag{4.20}$$

$$[\Delta x_k, \Delta y_k, \Delta z_k]\,, k = 1...n_{\text{points}} N_{\text{max.NN}} \tag{4.21}$$

The method getRotateZSDM() transforms the difference vectors according to a rotation with ψ and θ around the origin, see Equation 4.22. [105, 167]

$$\Delta z_k^{''} = -\sin(\theta)\Delta x_k + \cos(\theta)\sin(\psi)\Delta y_k + \cos(\theta)\cos(\psi)\Delta z_k \tag{4.22}$$

The transformed $\Delta z_k^{''}$ are counted into a one-dimensional histogram within the range of $\pm\Delta z_{\text{max}}$, the "z-SDM" (Equation 4.23). An example for a z-SDM from a selected precipitate with given ψ, θ can be seen in Figure 4.14.

$$\text{z-SDM} = \text{Hist}\left(\Delta z_k^{''}\right)|_{-\Delta z_{\text{max}}}^{\Delta z_{\text{max}}} \tag{4.23}$$

The method getRotateXYSDM(), additionally computes the transformed $\Delta x_k^{''}$ (Equation 4.24) and $\Delta y_k^{''}$ (Equation 4.25) difference vectors similar to Ref. [168]. The first rotation rotates ψ around the x-axis and the second rotation θ around the prior-gained $y^{'}$-axis (Figure 4.15), it should be noted that ψ and θ, so-defined, rotate in the mathematical "negative" sense. The method getRotateXYSDM() further takes a subset around a input $\Delta z_k^{''}$ position $\pm\Delta\Delta z$ and counts the difference vectors into a two-dimensional histogram, the "xy-SDM" (Equation 4.26). An example for the xy-SDM is shown in Figure 4.16 for an artificially created C14 crystal structure.

$$\Delta x_k^{''} = \cos(\theta)\Delta x_k + \sin(\theta)\sin(\psi)\Delta y_k + \sin(\theta)\cos(\psi)\Delta z_k \tag{4.24}$$

$$\Delta y_k^{''} = \cos(\psi)\Delta y_k - \sin(\psi)\Delta z_k \tag{4.25}$$

$$\text{xy-SDM} = \text{2D\_Hist}\left(\Delta x_k^{''}, \Delta y_k^{''}\right)|_{-\Delta x_{\text{max}}}^{\Delta x_{\text{max}}}|_{-\Delta y_{\text{max}}}^{\Delta y_{\text{max}}} \tag{4.26}$$

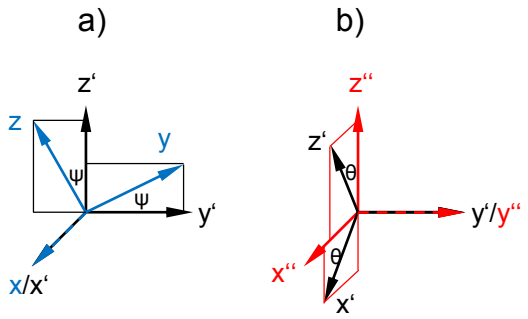

Figure 4.15: Defined rotations ψ in a) and θ in b).

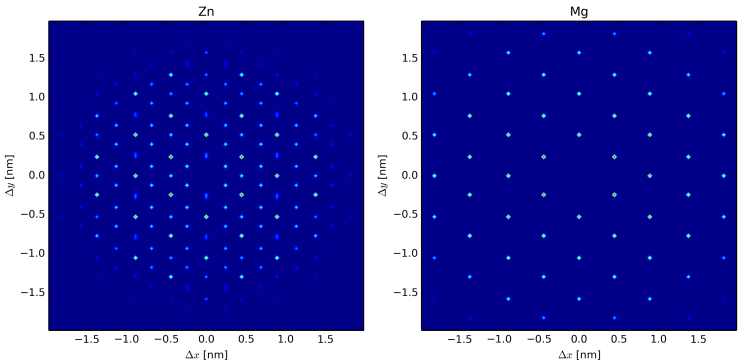

Figure 4.16: xy-SDM of an artificial C14 crystal ($MgZn_2$) for Zn-Zn and Mg-Mg. The z coordinate is parallel to $\vec{c}$ of the C14 crystal structure, $\Delta z = 0$ and $\pm\Delta\Delta z = 0.01$ nm.

With getRasterwinkel() a given set of atoms positions can be searched for a maximum of the z-SDM, scanning in a given (ψ, θ) space. In Figure 4.17 getRasterwinkel() was applied three consecutive times to find the (ψ, θ) for the maximum of the z-SDM of the selected subset containing the (002) pole.

Cluster search

A cluster search algorithm is implemented with the getClusterAtomsL() method (apt_importers.py, section 8.2). It is intended to represent the core-linkage method [119], but was implemented slightly different. As input the parameters $d_{\max}$, d_{link}, d_{erode}, K, the atom positions and possible "core" atoms are needed. The method identifies the possible core atoms with $d_{\max}$ and K as can be seen in Equation 4.27 [119].

$$d(A, A^K) \leq d_{\max} \tag{4.27}$$

Meaning that if the distance of an possible core atom A, to the K^{th} nearest possible core atom A^K is $\leq$ than $d_{\max}$, the possible core atom A is accepted as core atom and vice versa. Further all atoms (regardless if solute or not), within a range of d_{link} to the before identifed

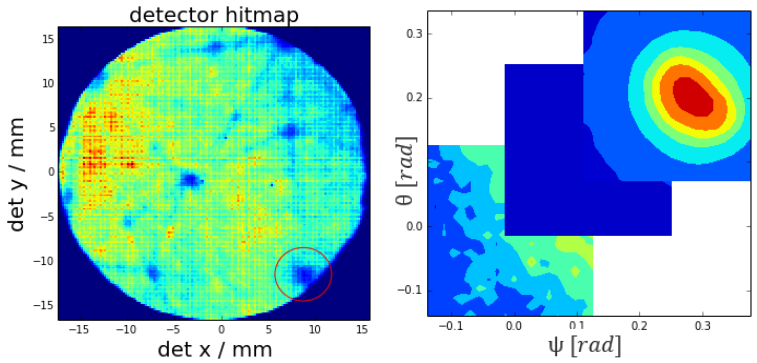

Figure 4.17: Selected subset of reconstruction, (002) pole, and maximum of z-SDM over ψ, θ, calculated with three consecutive runs of getRasterwinkel().

core atoms, are added to the "clustered atoms" set. In the core-linkage method now the cluster identity is assigned, all connected atoms (atoms within d_{link}, of the "clustered atoms" set) are assigned the same cluster identity. I.e. atoms which are farther from a identified cluster than d_{link} and belong to the "clustered atoms" set, will be assigned a different cluster identity.

However, the implemented method uses a different sequence: it erases previously all atoms from the set which are within d_{erode} from the matrix and then assigns the cluster identities. This can lead to different results from the core-linkage method if $d_{\text{erode}} > 0$. A cluster can be split up into several clusters this way by the erosion step, wheras in the core-linkage method only the number of the atoms in the cluster will be lowered. Therefore, if the method is used, it is used with $d_{\text{erode}} = 0$. Besides from this, there is also a difference between the core-linkage method and the maximum separation method (used by IVAS$^{\text{TM}}$). In the maximum separation method the assignment of the cluster identities seems to be done based only on the $d_{\max}$ and K parameter, and d_{link} and d_{erode} only adds or subtracts non-core (non-solute) atoms from the assigned clusters. Due to the de-facto only used method from IVAS$^{\text{TM}}$ for APT in literature, if comparisons to literature are needed the cluster-search analysis is done with the IVAS$^{\text{TM}}$ maximum separation method.

In general the linking/erosion approach is an unsatisfactory solution approach and seem to be arisen early, only due to the lack of a computationally effective method, for identifying points within a volume, which is only defined by other points. Therefore the cluster-search of Ref. [124] as used in clusteranalyse_AlMgSi.m (section 8.12) seems to be a more up-to-date approach for this issue.

For visual assessment of 3D data the volvis() method has proven to be an effective tool.

4.2.2.2 Analyses regarding the reconstruction protocol

Due to the use of reflectron-fitted atom probes in this thesis the flight path of L 382 mm cannot be used for the calibration of the ξ (ICF) value. We calculate a virtual flight path $L_{\text{virt.}}$ which can be used for ICF calibration as described in the following (analyse_recon.py, section 8.4).
A reconstruction is build with arbitrary k_f and ξ. The .epos file is exported from IVAS$^{\text{TM}}$ and the reconstruction is applied in a reverse manner with re-formulated Equations from section 4.2.1.6. The $X_i^{'}, Y_i^{'}$ coordinates are calculated with $L = 382$ mm, $F_e = 19$ V/nm, known k_f and ξ from the x_i, y_i coordinates of the reconstruction (Equation 4.28, 4.29, 4.30 and 4.31).

$$m = \xi - 1 \tag{4.28}$$

$$R_i = \frac{V_i}{F_e k_f} \tag{4.29}$$

$$X_i^{'} = \frac{x_i}{R_i} \frac{L}{m + \cos\left(\arcsin\left(\frac{\sqrt{x_i^2+y_i^2}}{R_i}\right)\right)} \tag{4.30}$$

$$Y_i^{'} = \frac{y_i}{R_i} \frac{L}{m + \cos\left(\arcsin\left(\frac{\sqrt{x_i^2+y_i^2}}{R_i}\right)\right)} \tag{4.31}$$

The $X_i^{'}, Y_i^{'}$ are compared to the X_i, Y_i coordinates of the .epos file and "magnification" values c are calculated thereof, for different detector positions of the reconstruction (Equation 4.32, 4.33, 4.34 and 4.35).

$$c_{1,3} = \frac{X_i}{X_i^{'}}\Big|_{max.X_i, min.X_i} \tag{4.32}$$

$$c_{4,5} = \frac{Y_i}{Y_i^{'}}\Big|_{max.X_i, min.X_i} \tag{4.33}$$

$$c_2 = \text{Mean}\left(\frac{X_i}{X_i^{'}}\right) \tag{4.34}$$

$$c_6 = \sqrt{\frac{A}{A^{'}}} \simeq \sqrt{\frac{(X_{\max} - X_{\min})(Y_{\max} - Y_{\min})}{(X^{'}_{\max} - X^{'}_{\min})(Y^{'}_{\max} - Y^{'}_{\min})}} \tag{4.35}$$

The virtual flight path is calculated via Equation 4.36.

$$L_{\text{virt.}} = cL \tag{4.36}$$

It should be noted that detector coordinates for reflectron-fitted instruments are first fitted with fitting functions, to correspond to straight flight path machines detector hitmaps, before the reconstruction protocol is applied [94], but in the .epos only the real detector hitmap locations are given and the tranformation functions are unknown. That is the reason why there is a difference for the magnification values over the detector area and the application of Equation 4.36 in computing a virtual flight path can only be seen as an approximation.

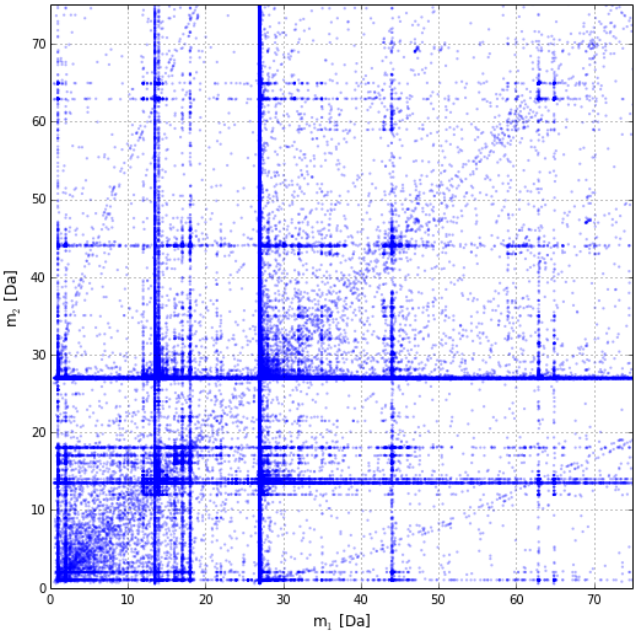

Figure 4.18: Ion correlation diagram for a voltage mode measurement.

However, using Equation 4.35 leads to consistent $L_{\text{virt.}}$ values of $\sim 44 \pm 0.5$ mm for different reconstructions. The usual good consistency of the measured, calibrated inter-plane spacings for three poles, supports the approach of a "virtual flight path" for the ICF calibration.

4.2.2.3 Other data analysis methods

For muliple ion analyis multiple_ion_analysis.py (section 8.6) is used. The aim is to plot a "Saxey plot [161]", also known as ion correlation diagram. Here all possible two-combinations of the mass-to-charge state (m_1, m_2) for all multiple events (ions per multiple event ≥ 2) are computed and m_2 over m_1 is plotted, due to symmetry only the two-permutations of (m_1, m_2) need to be calculated. A typical ion correlation diagram for a voltage mode run is seen in Figure 4.18. The number of permutations (i is the i^{th} multiple hit) to be calculated are shown in Equation 4.37.

$$n_{(m_1,m_2)} = \sum_i \binom{n_{\text{ions},i}}{2} \tag{4.37}$$

Specific patterns for in-flight dissociation of complex ions and the occurence of standing voltage evaporation can be obtained from this plot. [161] Element-specific detector hitmaps are generated in plot_multiple_hits_Si.py (section 8.3) starting from the epos file. Two-dimensional histograms are calculated from the detector coordinates (X_i, Y_j) for the respective subsets. The element-specifc detector hitmaps are used for artefact identification and as basis for cutting parts of the reconstruction. The detector hitmap in general is used for the calibration via identification of poles. If no poles are visible at the hitmap, the pulses

since last event pulse ("pslep") information can additionally be used to help make poles visible [169], but for Al alloys (generally they are low in alloying content, and poles are easily seen) this is usually not needed.

Chapter 5

Atom probe tomography study of as-quenched Al-Mg-Si alloys*,**

This study deploys a new method to gain insight into the as-quenched microstructure of Al-Mg-Si alloys using atom probe tomography (APT) as an imaging method. Here diffusion of solutes during sample preparation and handling is suppressed via application of cryogenic temperatures beginning from quenching in liquid nitrogen (LN_2) through to APT experiments at 33 K. The solute distribution is studied via customized nearest-neighbor distribution and radial distribution function analysis. The influence of energy input on the solute distribution via cryogenic focused ion beam (FIB) preparation is also shown.

*Chapter 5 was already published in [144].

**This research is supported by the Austrian FFG Bridge project, number 853208. The authors want to thank Francisca Mendez Martin and Katharina Babinsky, Department of Physical Metallurgy and Materials Testing, Montanuniversitaet Leoben, for the introduction to the reconstruction process with software IVAS^{TM}. Peter Felfer, FAU Erlangen, is kindly acknowledgedthanked for permission to use his MATLAB analyzing scripts and the online tutorial on how to use them.

5.1 Introduction

Al-Mg-Si-(Cu) alloys are a widely used material in various products which are manufactured in rolling, extrusion, forging and drawing processes. [1] The main strengthening mechanism used to enhance the mechanical properties in this alloy class is precipitation strengthening. The heat-treatment pathway is crucial for the in-application properties of these alloys. [38] The amount of time the material is held or stored at room temperature (RT) after solution heat-treatment and quenching can have a negative influence on its mechanical properties if the material is artificially aged (AA) afterwards at temperatures of typically 453 K. This "negative effect" due to natural aging (NA) lowers the hardness and yield stress of the material and reduces artificial aging kinetics. Nowadays it is generally accepted that the "negative effect" is linked to solute clusters formed at room temperature. [44] It has also been deduced that quenched-in vacancies play an important role in nucleation of the clustering process during NA. However, targeted alloying and processing can be deployed to delay NA response and to improve the AA response. [12, 41, 44, 45] Indirect methods such as positron annihilation spectroscopy (PAS), differential scanning calorimetry (DSC) [170], electrical resistivity measurements [64] and hardness measurements are frequently applied to gain insight into clustering kinetics. [1] For NA different stages have been determined, ranging from I to IV, where different mechanisms are assumed to predominate. [88] Note that the kinetic details depend on the exact alloy composition and not all stages can be observed clearly in every alloy experimentally. In the first stage vacancy annihilation, vacancy-solute pair formation, vacancy-cluster -formation and build-up of Frank loops are thought to be the dominating processes. Stage II may be dominated by Si cluster formation. It has been proposed that Mg starts to contribute to NA during stage III. In stage IV cluster growth takes place. [45, 53] The latest results on time-dependent magnetization also indicate Si clustering in early stage II due to the Si-dependent activation energy of the stage II/III transition. [91] Nevertheless, the role of Si before Mg gets involved is still under debate, because this stage is difficult to access experimentally in Al-Mg-Si-(Cu) alloys. Diffraction methods such as small-angle X-ray scattering (SAXS), small-angle neutron scattering (SANS), and diffraction patterns in transmission electron microscopy (TEM) suffer from a low signal-to-noise ratio caused by the similar scattering factors (or electron densities) of the periodic table neighboring elements Mg, Al, and Si. [4] As a direct imaging method, atom probe tomography is used to study the influence of different heat-treatment states on clustering and the role of additional alloying elements. [129, 133, 138] The characterization of clusters and their linking of their properties to mechanical properties can also be realized. [171] However, measuring within the first hours after quenching has not been possible, because the APT method suffers from the amount of time needed for sample preparation and sample transfer times in the analysis chamber at ultra-high vacuum conditions. [88] In this study we demonstrate that by applying a new sample preparation and manipulation method, direct APT observation of

Al-Mg-Si-(Cu) alloys after quenching is possible. We also present preliminary results on very early-stage decomposition.

5.2 Experimental

APT samples were prepared from alloy AA6016. The composition of the material was measured with a spark optical emission spectrometer and determined to be (in at.%): 1.0 % Si, 0.4 % Mg, 0.03 % Cu, and 0.005 % Ga. Typical APT sample blanks of 1×1 mm alloy were electropolished to a pre-tip radius of approximately 20 μm. The large radius of the pre-tip was designed taking into account Einstein's diffusion distance for Mg at 803 K to hinder preferential loss of Mg in our final APT tip, which was then prepared via FIB in the center of the needle. The pre-tips and Mg, which served as getter material, were placed in capsuled quartz tubes (∼150 mm in length and ∼10 mm in diameter) which had been repeatedly evacuated and purged with high-purity argon (5N) to a pressure of ∼300 mbar. The encapsulated pre-tips were solution heat-treated at 803 K for 5 min in a furnace and quenched to the temperature of LN_2 by plunging the capsules into LN_2. The plunged quartz capsules were cracked and opened under LN_2 to access the samples. The samples stored at LN_2 were then dipped into ethanol at RT for handling purposes and to remove moisture contamination, before inserting them into a Leica vacuum cryo specimen transfer shuttle, which was rapidly pumped down to 10^{-5} mbar usingvia a Bal-Tec BAF060 freeze-etching chamber. The sample was subsequently vacuum-cryo-transferred to a pre-cooled FEI FIB-SEM Helios 600i stage at 123 K. Note that the handling procedure involved approximately 1 min at RT. Even this short time at RT may be omitted in the near future via total handling under LN_2. Final cryo-FIB sample preparation involved cutting back the pre-tip by approx. 20 μm (to reach a region where no Mg is lost due to solution annealing). The APT tip was produced via standard initial 30 kV annular milling and a final 5 kV cleaning. A schematic overview of the whole procedure is illustratedcan be seen in FIG. 5.1. The APT tip was then vacuum-cryo-transferred via the VCT shuttle to a modified LEAPTM 4000X-HR atom probe system; the VCT shuttle which enables cryogenic sample transfers into the atom probe analysis chamber [149, 172], where the specimen then resides at 33 K. The samples were analyzed in voltage pulse mode with a pulse fraction of 0.2, a pulse frequency of 200 kHz and a detection rate of 0.5 % at a specimen temperature of 33 K. The reconstruction of APT datasets was performed wiusingthin IVASTM 3.6.10. To calibrate the reconstruction an advanced method was used, which is briefly explained as follows [109]. During an APT experiment, faceting of the tip can occur for stable planes of the crystal, which can be observed as low relative hit density regions on the detector hit map. These are called crystallographic poles because the specific planes are normal to this direction. Here spatial distribution maps (SDMs) [107] are used to quantify the atomic plane distance for

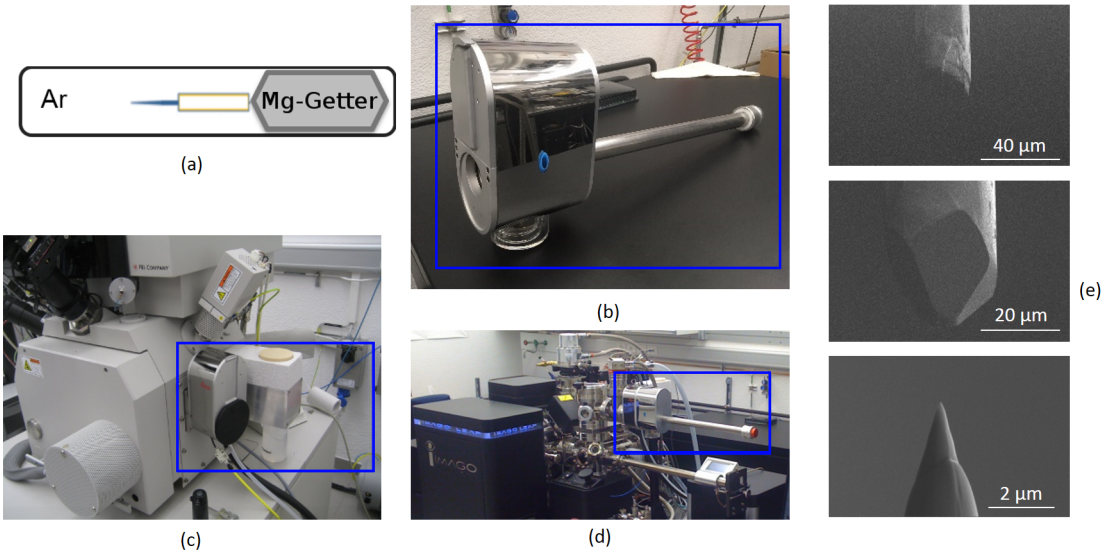

Figure 5.1: a) Solution heat-treating of the pre-polished samples. b) Leica vacuum cryo specimen transfer shuttle (VCT100). c) FEI FIB-SEM Helios 600i, Cryo-FIB. d) Specimen (vacuum/cryo) transfer to atom probe. e) Evolution of the tip during cryo-FIB operation. Cryo-sample preparation and transfer procedure:. Pre-polished samples are solution heat-treated and quenched in LN_2. Subsequently they are inserted into the cryo-transfer shuttle and the APT specimen is cut in the cryo-FIB. Due to Mg loss during heat treatment severe cutting of the pre-electropolished tip is required. The transfer from the cryo-FIB is implemented again via VCT shuttle to the atom probe. The parts framed in blue in b), c), d) are the transfer shuttle and the corresponding docking parts.

different orientations to optimize input parameters, i.e. the field factor (kf) and the image compression factor (ICF), to ultimately calibrate the reconstruction. The (200), (220) and (311) poles were used to calibrate the reconstruction to their respective interplanar spacings, according to the fcc lattice parameter of Al (unit cell, 0.405 nm). Further, data treatment was realized using customized scripts within python [173], numpy [174], the python binding to the FLANN library [166], matplotlib [175], MATLAB [165], and scripts from [164] and [124] in interaction with blender [176] and VisPy [177]. In the vicinity of poles artifacts are known to occur. [113, 115] This is especially critical in the case of Al-Mg-Si alloys, where Si has been shown to migrate towards or away from specific poles during APT measurements. [114] To account for this in a reproducible manner, a customized pole and surface identification routine was applied. [119] Based on the respective 100^{th} nearest-neighbor distance, a local density value wais computed for all given atoms. [119] A density of 14 atoms nm^{-3} was applied as a threshold value, and the low density artifacts (indicating crystallographic poles) were thereby removed. Atoms located within a distance of 2 nm from such identified atoms were also removed. [119]

5.3 Results

To characterize the solute distributions we used the 10^{th} nearest-neighbor distance distributions and a measure as defined in Equation 5.1 and Equation 5.2, where (R_i) is the position vector of a specified solute atom. This defined measure is comparable to a radial distribution function, or pair correlation function (Equation 5.1) as used in [138, 140]. Random comparator curves are generated through random labeling of the existing atom positions. The random samplings of the specified number of atoms are drawn 40 times from the set and the nearest-neighbor distribution and the so defined "RDF" are calculated and averaged. The standard deviation is calculated from the various random distributions drawn.

$$\text{RDF} = \text{Hist}\left(\left|\left|\vec{P}_i - \vec{P}_j\right|\right|_2\right) \Leftrightarrow i \neq j \tag{5.1}$$

$$\text{ratio}(r) = \frac{\sum \text{RDF}}{\sum \text{RDF}_{\text{rand}}} \tag{5.2}$$

In the following we discuss two different cases, a successfully measured as-quenched (AQ) condition and an as-quenched specimen which was prepared under FIB conditions, resulting in Ga implantation (0.6 % Ga on average) denoted here as "Ga implanted". All runs showed a yield of around 3×10^6 collected ions. For the AQ sample the Si-Si spatial distribution shows no significant difference from a random solid-solution case, as can be seen in FIG. 5.2 a) and b). A slight trend towards Si-Si aggregation may be deduced from FIG. 5.2 b), but the error bars partially overlap unity. Note that a value >1 indicates a non-random solute distribution and that the position of the drop-off to unity is always larger than the expected cluster radius, if we assume that clusters are surrounded by regions of lower solute concentration, which is inherent in the definition of clustering. For the Mg-Mg spatial distribution in FIG. 5.2 c) and d) clearly no significant difference from a random solid solution is observed. Finally, a cross Si-Mg spatial distribution as presentedseen in FIG. 5.2 e), f) also shows no significant aggregation of Mg to Si atoms. For this case the comparator was built by fixing the Si atom positions and randomly labeling the Mg positions on the remaining non-Si positions. Note that qualitatively the results are the same if the Mg atoms are fixed.

FIG. 5.3 a) shows the results for the distribution of the solute Si in a freshly quenched state in the "Ga implanted" sample. Obviously the distribution of Si atoms is now discernibly different from that in the random sampling. Even better visibility is provided by the ratio plot in FIG. 5.3 b). The Mg-Mg spatial distribution as shown in FIG. 5.3 c), d), however, shows no significant differences from a random solid-solution distribution: FIG. 5.3 c), d). In addition, no cross Si-Mg aggregation wascould be obtained; see: FIG. 5.3 e), f). Note that the Ga concentration was, as expected for FIB-prepared APT samples, not uniform over the specimen and more Ga atoms are located near the surface of the specimen. Excluding this region near the surface from APT data, would, however, not influence the analysis in FIG. 5.3 in a relevant manner. Applying a cluster search algorithm similar to that described

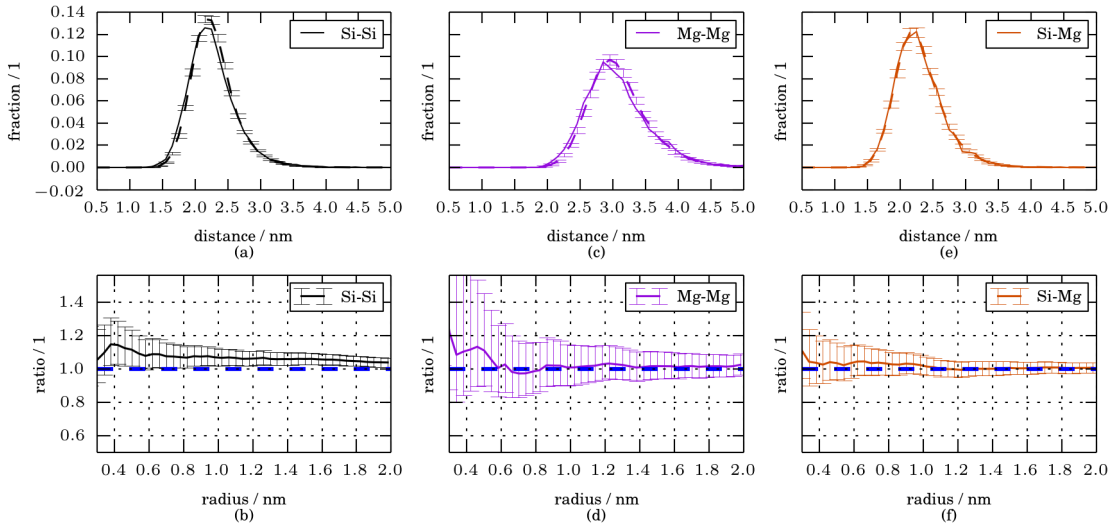

Figure 5.2: As-quenched sample: a), c), e) 10^{th} nearest-neighbor distance distribution of Si-Si, Mg-Mg, and Si-Mg. Results for the nearest -neighbor distribution are normalized. b), d), e). The ratio as defined in Equation 5.1 and Equation 5.2. Error bars show the standard deviation of the random labeling results. To calculate the "ratio" error bars the limiting curves from the standard deviation of RDF_{rand} were used and processed as in Equation 5.2. Si-Mg spatial distributions are calculated for fixed Si positions. The dashed blue line indicates the reference for a random distribution. No significant Si-Mg or Mg-Mg aggregation can be determined.

in [119] and [124] showed that the Si clusters identified are located within the bulk of the sample. Here the results for the "Core-Linkage" algorithm for Si, Mg and Cu as possible core atoms with the parameters $d_{\text{max}} = 0.74 = d_{\text{link}} = d_{\text{erode}}$, and K=5 are discussed. If we only consider the Si, Mg and Cu atom coordinates we calculate a mean Guinier radius [94] of 0.7± 0.1 nm. The inter-cluster distance is 6.8±1.9 nm. It is important to note that Ga-free APT tips can certainly be produced via FIB. However, one needs to be careful and take into account the energy transfer into the system when early clustering is studied in as-quenched alloys.

5.4 Discussion and conclusion

The as-quenched state of Al-Mg-Si-(Cu) alloys shows no clustering of Si-Si, Mg-Mg, and Si-Mg solutes, even though Si-Si clustering is at the significance level (FIG. 5.2). Unfortunately, only a low number of atoms could be obtained during these proof-of-principle experiments, and this number is reduced further due to the removal of artifact-filled data in the vicinity of poles. This and low solute concentrations generate large error bars in the resulting solute characterization measures, which we will try to improve in future mea-

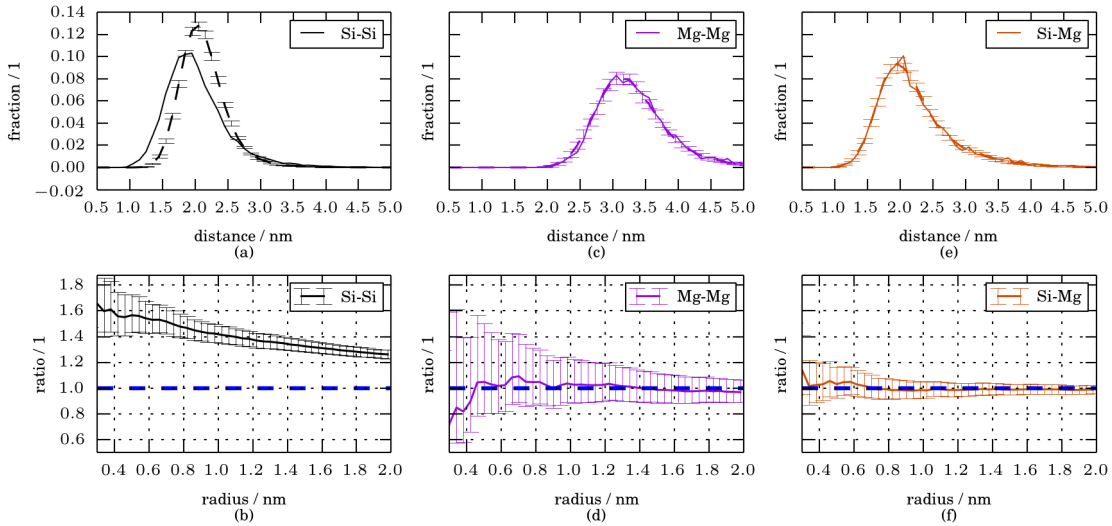

Figure 5.3: Ga-implanted as-quenched sample: a), c), e) 10^{th} nearest-neighbor distance distribution. Results for the nearest-neighbor distribution are normalized. b), d), e) The ratio for Si-Si, Mg-Mg, and Si-Mg, respectively. Error bars show the standard deviation of the random labeling results. For the calculation of the "ratio" error bars the limiting curves from the standard deviation of RDF_{rand} were used and processed as in Equation 5.2. Si-Mg spatial distributions are calculated for fixed Si positions. The dashed blue line indicates the reference for a random distribution. Only significant Si-Si aggregation can bewas obtained from a) and b), and neither Mg-Mg aggregation ((c), d)) nor Mg to Si aggregation can be obtained ((e), f)).

surements. However, the Ga-implanted sample clearly shows Si aggregation (FIG. 5.3). In this sample the energy input due to Ga-implantation is expected to induce clustering, even at 123 K. As a rule of thumb, Ref. [95] states that for 0.1 % Ga in a Fe sample, 0.75 displacements per atom are calculated using Ref. via [178]. Due to the fact that we have a higher Ga-implantationementation in Al, with a lower molar mass and Young's modulus compared to Fe, we expect a much greater number of displacements per atom and many non-equilibrium vacancies [179], which accelerates diffusion. [44] Currently no APT measurements for the as-quenched state are available in the literature, due to the experimental limitations described in the introduction. Nevertheless, samples naturally aged for a short time (due to sample preparation and handling limitations) are sometimes inaccurately termed "as-quenched". The earliest of such measurements, after approximately one hour of natural aging, reveal Mg-Mg and Si-Si, but no Si-Mg correlations. [138] Comparing our results of the freshly quenched specimens to the indirect measurements via positron annihilation lifetime spectroscopy in [48], we are investigating samples at stage II of natural aging. Stage II is characterized by a drop in the average positron lifetime over natural aging

time, which is attributed to vacancy annihilation and the commencement of Si cluster formation [45, 48, 53]. For the as-quenched state, this study found no significant Mg-Mg, Si-Si or Si-Mg aggregation. However, Si-Si clustering is at the significance level, which fits stage II of RT clustering in Al-Mg-Si alloys. Further insight into the early stage of low-temperature clustering is possible from our results on the significant Si-Si aggregation in Ga-implanted samples. We propose the clustering of Si to be caused by the vacancies introduced during FIB preparation. Because the vacancy concentration directly effects clustering kinetics [44], high mobility is possible even at 123 K. The fact that only Si starts to form clusters and Mg is not involved in this process is the first direct evidence that Si, as long suggested, is the most mobile species during the early-stage decomposition of quenched Al-Mg-Si-(Cu) alloys. In summary we show here that it is possible to study the as-quenched super saturated solid-solution state in Al-Mg-Si-(Cu) alloys via cryo-transfer enabled atom probe tomography, and that Si is likely to be the first solute involved in low-temperature clustering.

Chapter 6

Size Dependent Diffusion: Material Dimensions Determine Solid State Reactions*,**

The key question in material sciences is how fast properties evolve, which implies kinetics of phase transformations. In metals, kinetics is primarily connected to diffusion via atomic lattice vacancies and often non-equilibrium vacancies are required for structural changes. For example, rapid quenching of various important alloys results in natural aging, i.e. slight movements of solute atoms in the material, which significantly alter the materials properties. In this study we demonstrate that all solid state reactions based on non-equilibrium substitutional diffusion are size dependent. We illustrate the size effect on clustering in an aluminum alloy via an imaging method with near atomic resolution, i.e. atom probe tomography. We show that the diffusional process is effectively stopped when the sample size reaches the nanometer scale, a fact which (beside its technological and academic importance) also has huge implications for the study of non-equilibrium diffusion and microstructural changes via microscopic techniques.

*Chapter 6 is in revision for publication in [180].

**This research was supported by the Austrian FFG Bridge project, number 853208. AMAG Rolling GmbH is thanked for financial support and discussions. This project also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant No. 757961).

The authors want to thank Mr. Camenzind, Mr. Rechsteiner and Mr. Eisenhut from EMPA, and Mr. Jaggi and Mr. Schneebeli from SLF Davos for organization and help in using their arctic chambers. We also kindly thank Mr. Kollender from JKU for his recommendation of an electrolyte suitable for electro-polishing at low temperatures; Mr. Bartelme of Montanuniversitaet Leoben for his help with the sample production in the arctic chamber; and Ms. Mendez Martin, Montanuniversitaet Leoben, for organizing APT measurements.

6.1 Introduction

The kinetics of phase transformations is a central topic in material science. Frequently, non-equilibrium vacancies which are induced via rapid cooling, irradiation, sputtering or plastic deformation [41, 181–183] are required to activate structural changes. Already in 1911 hardening during room temperature storage of Al alloys was accidentally discovered by Wilm when trying to harden an Al alloy like steel by quenching. Later this effect was given the name natural aging (NA) [14]. The hardness increase during room temperature is attributed to the formation of a nanometer-sized unordered accumulation of solute atoms in the material, so-called clusters. Kinetics of NA strongly depends on non-equilibrium vacancies. Today the effect has huge importance for all classes of novel high strength aluminum alloys [3]. Lately it has attracted more and more interest, also for magnesium alloys [184–186] due to the improvement of characterization methods, i. e. microscopic techniques with atomic resolution [36, 106, 187, 188]. An aluminum alloy in which natural aging has been studied intensively over the past 20 years [7] is type AlMgSi, where a detrimental effect of NA on mechanical properties [2] is observed and limits its extended use in various areas of lightweight applications [44]. (Note that NA can be also beneficial for gaining high strength in other alloys [11].) In Fig. 6.1 we illustrate the complex effect of natural aging in an AlMgSi

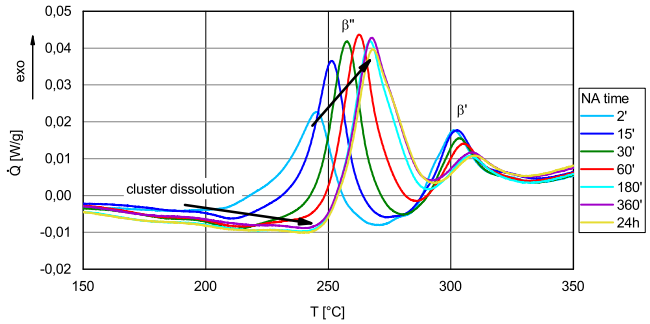

Figure 6.1: Excess heat flow per sample mass (DSC heating curves) with varying natural aging time after quenching of an AlMgSi alloy. Even for short NA the cluster dissolution increases and the formation of the main hardening precipitate (β'') is hindered upon heating.

alloy (EN AW 6016) via differential scanning calorimetry (DSC). Even for short times after quenching, the AlMgSi alloy shows a pronounced change in DSC traces. With increasing NA time enhanced cluster formation (see supplementary material Fig. 6.5) results in increasing endothermic cluster dissolution upon heating. The formation of the main hardening phase (β'') is retarded [70], indicating the negative effect of NA in this AlMgSi alloy.
Since the emergence of atom probe tomography (APT) a number of studies involving the direct observation of clusters in Al alloys have been conducted [6, 7, 73, 129, 131, 135, 138]. However, they have generated contradictory results with regard to the sequence of cluster formation, especially concerning the early stages of clustering [7, 73, 131, 135, 138]. [3] In

the following we show that the disagreement may be caused by an incorrect assessment of the NA time.
Usually the NA time is determined by the total time samples or components experienced at room temperature after quenching, Δt_{NA}. Often an explicit distinction between the NA of finished APT samples or bulk material is lacking. Here we demonstrate that the above definition of Δt_{NA} is not generally valid and that the amount of clustering is governed only by the time during which the material is exposed at bulk dimensions. This is seen as a universal effect and is not limited to the example alloy.
We illustrate this "problem" by means of two differently designed experiments. The first uses "nano tip aging" where the NA is performed *in-situ* in the atom probe on finished nano-sized APT tips. In the second experiment we perform "bulk aging" where the NA time at bulk dimensions is varied, but the "total NA time" (the sum of "bulk aging" and "nano tip aging") is kept constant.

6.2 Results

6.2.1 Nano tip aging

For "nano tip aging" sample preparation and transfer took place under arctic conditions and a special atom probe equipped with a novel cryo-transfer system [144, 188, 189] is used to suppress any diffusion during preparation and manipulation (see Methods section for details). Fig. 6.2a provides an overview of the in-situ sample "nano_tip_aged_01". To our surprise and excitement, spatial analyses (observed distribution in comparison to a random solid solution; see Methods section for details) of the solute species Mg and Si, shown in Fig. 6.2b, reveal no significant effect of in-situ aging for the Mg-Mg, Si-Mg, Mg-Si or Si-Si (see supplementary material Fig. 6.6 and "Si migration/surface relaxation and regions of interests" for details). Obviously the solute distribution stays completely random over the applied NA time, Δt_{NA}, up to three weeks. This result is contrary to all expectations and literature results on NA, also to those from DSC in Fig. 6.1, where microstructural changes were already obvious after several minutes of Δt_{NA}. An explanation for this unexpected result follows. Because clustering upon NA is in general a substitutional diffusion process at room temperature, it only happens due to the availability of non-equilibrium vacancies from quenching [41].

6.2.2 Vacancy annihilation

In Fig. 6.3 we show the results of thermo-kinetic calculations which we conducted for a thermal route similar to the applied in-situ sample processing in order to quantify the non-equilibrium vacancy fraction. The diameter of a sphere, synonymous to the maximum dis-

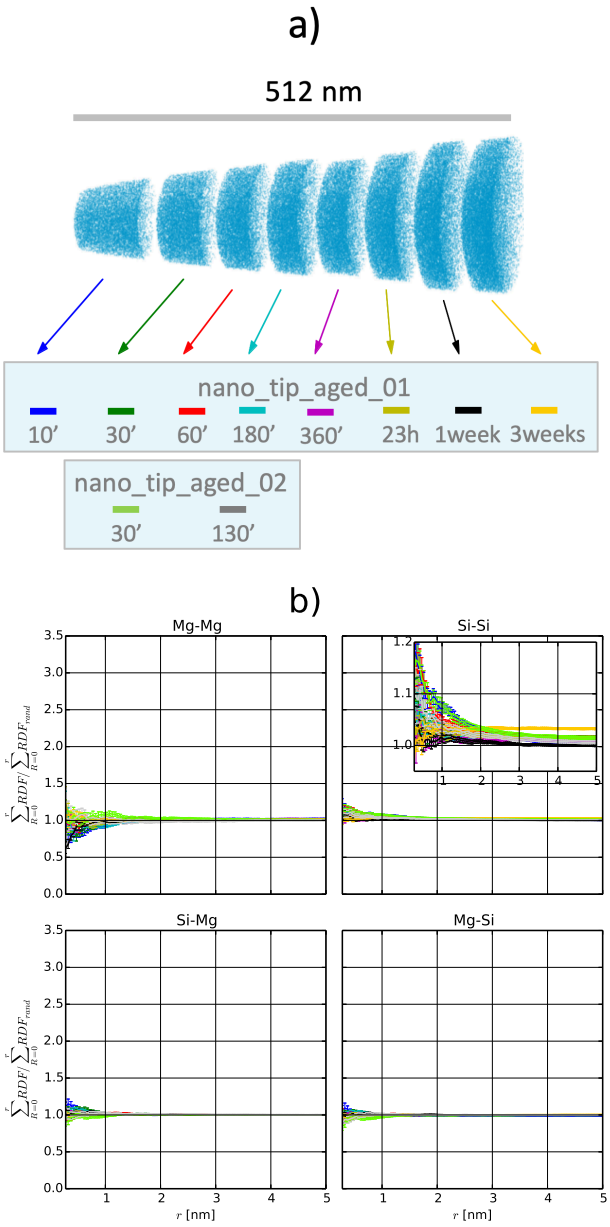

Figure 6.2: In-situ nano-tip natural aging of atom probe samples of quenched AlMgSi alloy. a) Mg atom positions of the concatenated runs of the sample "nano_tip_aged_01" (always shifted by 5 nm in z direction from the max(z) of the previous run). b) Analysis of the spatial positions of solute atoms: Shown is the ratio of the cumulative sums of the radial distribution function (Equation 6.1 in the Methods section) for the given interactions (Mg-Mg, Si-Si (insert magnified view), Si-Mg and Mg-Si) for the "nano_tip_aged" samples. Values > 1 for the ratio indicate clustering (for details see Methods section and supplementary Table 6.1). No deviation from random can be observed for all natural aging times Δt_{NA}, a result which was completely unexpected and contradicts literature on natural aging.

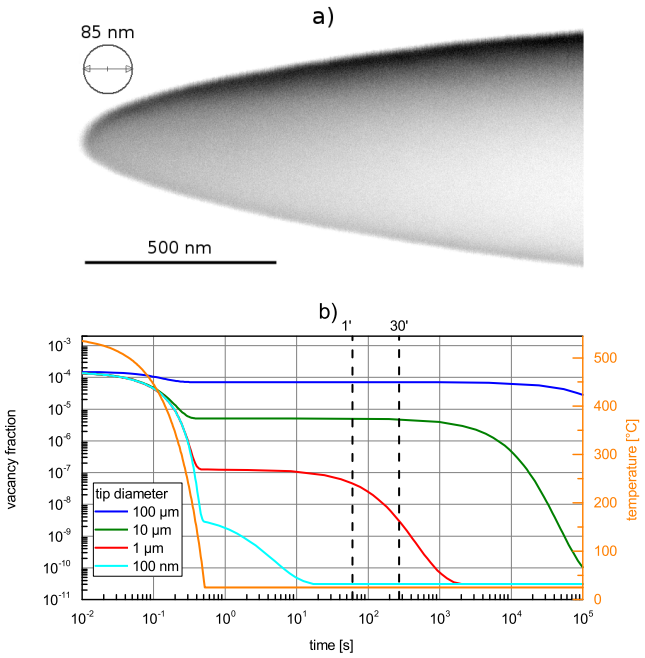

Figure 6.3: Simulation of the non-equilibrium vacancy evolution. a) Inverted scanning electron microscopy image of the APT sample "nano_tip_aged_01" before measurement. b) Calculated non-equilibrium vacancy fraction over time and temperature upon quenching and natural aging for pure Al (FSAK model [42]). A sphere diameter is used as a simplified model for the nano tip. The non-equilibrium vacancy fraction formed upon quenching and its preservation at RT decays rapidly with decreasing dimensions. Additional lines for 1' and 30' are added as visual guidelines.

tance to the next vacancy sink, serves as a model for the APT tip, which is shown in Fig. 6.3a. The vacancy fraction calculated for pure Al is shown in Fig. 6.3b. Changes in the tip diameter largely influence the evolution of the non-equilibrium vacancy fraction: The frozen-in non-equilibrium vacancy fraction upon quenching is orders of magnitude lower if the tip diameter is decreased by an order, which means that the creation of a vacancy supersaturation is even difficult at small scales. Further, the decline to the equilibrium vacancy fraction is much earlier, and for a diameter of 100 nm, a size in the range of the APT tip radius, it is already reached in less than a minute. This suggests that the non-equilibrium vacancy driven process of clustering must be strongly size dependent and is suppressed at small dimensions.

6.2.3 Bulk aging

To further prove that clustering and NA are really stopped in nano-sized APT samples, we conducted the second APT experiment. Here the sum of "bulk aging" and "nano tip aging" was kept constant. The time during bulk NA ("bulk-aging") was varied, but the total NA

time Δt_{NA} of the APT samples was preserved (illustrated in Fig. 6.4a). The APT sample "bulk_aged_01" was prepared 9 minutes after quenching the bulk (rods of 0.7 mm thickness) and then stored at RT as nano-tip for 3 weeks. The "bulk_aged_02" sample was prepared after 1 week of bulk-aging of the quenched rods and the nano-tips then further stored for 2 weeks. Fig. 6.4b shows the data obtained. If the definition of Δt_{NA} would be applied, no significant difference between the two measurements would be discernible because the time at RT after quenching is 3 weeks for both runs. However, the two states differ very clearly in the signal for clustering, in fact for all interactions Mg-Mg, Si-Si, Si-Mg and Mg-Si. Replicate measurements for the same bulk NA time, but other nano-tip NA time (in supplementary Fig. 6.12) show results almost identical to those presented in Fig. 6.4.

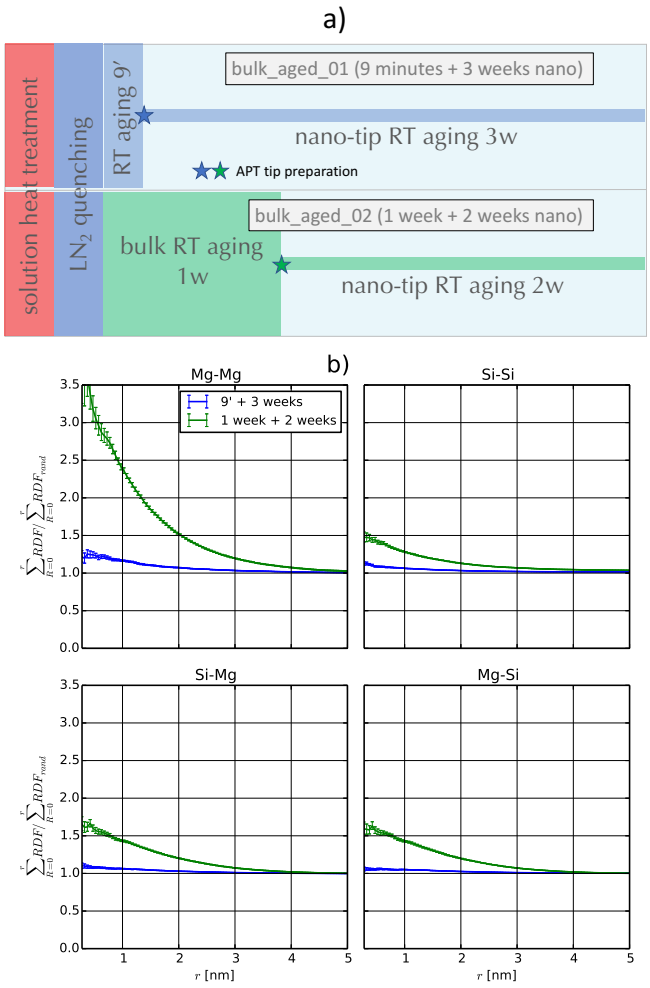

Figure 6.4: Bulk natural aging of quenched AlMgSi alloy. a) The schematic overview of the heat treatment and sample preparation procedure illustrates that a the total time after quenching is kept constant at three weeks, while only the point of preparation of the nano-sized atom probe tip is varied (9 minutes and 1 week). b) Analysis of the spatial positions of solute atoms (for details see Methods section and supplementary Table 6.1). Blue lines correspond to the sample "bulk_aged_01", green lines correspond to the sample "bulk_aged_02". A very pronounced difference can be observed although the total time after quenching Δt_{NA} is similar, but only the time at bulk dimensions has been varied. For the long natural aging the expected strong solute clustering upon natural aging is obvious.

6.3 Conclusion

We have shown that clustering of solutes after quenching in metals, as a diffusional process, not only depends on the storage time at room temperature, as is known in natural aging for more than 110 years, but also shows a pronounced size effect. The non-equilibrium diffusion effectively stops very quickly when the sample size approaches the nanometer range. Moreover, for small dimensions it is also impossible to reach a significant fraction of non-equilibrium vacancies upon rapid quenching, because vacancies, as the main carriers of non-equilibrium substitutional diffusion in metals are annihilated at the free surface of nano-sized samples during quenching and storage at room temperature. This is demonstrated clearly in simulations and experimentally. Our findings permit several general statements regarding metal alloys.

- All non-equilibrium substitutional diffusion-controlled processes are size dependent. They are strongly suppressed at small dimensions, regardless of how non-equilibrium vacancies are created, by thermal quenching or other means.
- The size dependency has to be considered when in-situ high resolution microscopy techniques such as transmission electron microscopy or atom probe tomography are utilised to study non-equilibrium kinetics in bulk materials.

Our findings also have specific implications for results in literature concerning the observation of early stages of clustering upon natural aging, especially in the field of aluminum alloys.

- All results of prior applied characterization methods which use small sample sizes may have been influenced by ill definition of the applied natural aging time.
- This applies particularly to atom probe tomography results over the last 20 years, and may explain differing results on short natural aging. For larger time spans the results are likely to hold, because such samples are usually processed contemporaneous with measurement in atom probe tomography.

6.4 Methods

Experimental parameters, data analysis and calculation parameters are briefly described in the following:

- **Material:** A commercial 6016 Al alloy with a nominal composition [at.%]: Mg 0.35%, Si 1.04%, Cu 0.04% and Al balance; measured via a spark optical emission

spectrometer, was used for all investigations.

- **DSC** (Fig. 6.1): The material was cut into samples and ground to a final mass of approximately 42 mg. The samples were heat treated at 545° C and quenched into LN_2. For each NA time three samples were measured against a high purity Al reference of the same mass, using Al crucibles and a heating rate of 10 K/min. The three curves obtained were shifted to zero at the solution heat treatment regime and the mean computed. Measurements were carried out on a Netzsch DSC 204 F1.

- **APT sample preparation** (Fig. 6.2, 6.4). "Nano tip aging" (see also supplementary Table 6.1): The cut blanks ($1 \times 1 \times 20$ mm) were first-step electro-polished (first-step, 25 % HNO_3 in methanol). Then a neck was micro-polished (second-step, 2% $HClO_4$ in 2-butoxyethanol) near the apex, with a diameter in the order of 5 to 20 μm. The necked samples were then solution heat treated at 545° C with N_2 purging and quenched into LN_2 and transported to the arctic chamber, where the micro-polishing was completed at -40° C (3% $HClO_4$ (72%), 16% 2-Ethoxyethanol, 22% 1,2 Dimethoxyethan in methanol). LN_2-cooled samples were dipped into room temperature ethanol beginning the natural aging for the respective time (10' and 30' for nano_tip_aged_01 and nano_tip_aged_02). Samples were then put into the cryo specimen transfer shuttle. The shuttle was fast pumped down to 10^{-5} mbar and the samples cooled to -120° C using a Bal-Tec BAF060 freeze-etching chamber. Subsequently, samples were vacuum-cryo transferred [189] to a pre-cooled FEI FIB-SEM Helios 600i and then to the analysis chamber of the APT. Additional NA times for the same sample were realized by stopping the run and transferring the sample to the buffer chamber (RT) and holding it. The respective time was added to the previous NA time.
  "Bulk aging" samples (see also supplementary Table 6.1): The cut blanks ($0.7 \times 0.7 \times 20$ mm) were solution heat treated at 545° C in an air furnace with N_2 purging and quenched into LN_2. "Bulk_aged_01" (Fig. 6.4a) was taken out of LN_2 and rapidly first- and second-step electro-polished within 9' at RT. "Bulk_aged_02" (Fig. 6.4a) and "bulk_aged_03" were taken from LN_2 and plunged into iso-propanol, stored for 1 week at room temperature, and first- and second-step electro-polished. Finished samples were again stored at room temperature for 2 weeks / 1 day respectively until APT measurement.

- **APT measurement parameters** (Fig. 6.2, 6.4): Samples were run in voltage mode with a pulse fraction of 20 %, 200 kHz and a detection rate of 1% at a temperature of 30 K. "Nano tip aging" samples and the sample "bulk_aged_01" were run on a LEAP 4000 X HR equipped with self-constructed cryo-transfer capabilities, and

"bulk_aged_02" and "bulk_aged_03" on LEAP 3000 X HR.

- **APT Data analysis** (Fig. 6.2, 6.4): For APT solute analysis the $^{24}Mg^{2+}$, $^{25}Mg^{2+}$, $^{26}Mg^{2+}$; $^{28}Si^{2+}$, $^{29}Si^{2+}$, $^{30}Si^{2+}$; and $^{24}Mg^{+}$, $^{25}Mg^{+}$, $^{26}Mg^{+}$ peaks were used. The reconstruction was built by calibrating the k_f and the ICF value as described in [109] within the commercial program IVAS 3.6.12. Artefacts due to pole migration [114, 190] were minimized by choosing regions of interest, neglecting data of (111) poles and the zone line (111)-(022). The defined regions of interest based on the detector hitmaps can be seen in supplementary Figs. 6.7, 6.8, 6.9 and 6.10. We use the ratio of the cumulative sums of the radial distribution function as depicted in symbolic form in Equation 6.1 [144] as a measure for clustering. The exact used definitions are discussed in the supplementary material section 6.6.2. Values > 1 for the ratio indicate clustering. This formalism has the advantage of being parameter-free [191], in comparison to a cluster finding algorithm, and still compresses the information from the whole spatial distribution of chosen solutes within a given radius r.

$$f(r) = \frac{\sum \mathrm{RDF}_{AB}}{\sum \mathrm{RDF}_{AB,\,\mathrm{rand}}} \tag{6.1}$$

- **Vacancy kinetics calulation** (Fig. 6.3): A thermokinetic calculation based on the FSAK model [42] which takes excess vacancies into account was computed via MatCalc 6. Pure Al was used as material and the sphere diameter was varied, as a model for the tip diameter and vacancy sink. A temperature history of: 545° C cooled with 1000 K/s to 25° C, and further natural aging at 25° C were applied. The other parameters were chosen as: dislocation density 10^{11} 1/m^2, jog fraction 0.02, Frank loop nucleation constant 0.0, jog fraction on Frank loops 0.2, Frank loop intf. energy 1.0, effective loop line energy $1/2Gb^2$ and excess vacancy efficiency 1.0 as used in Pogatscher et al. [41].

6.5 Contribution

P.D., S.P. and P.J.U. conceived the study. P.D. and P.J.U. produced samples. P.D. and S.S.A.G. did the measurements. P.D. and S.P. did the calculations. P.D. did the data analyis. S.P., P.J.U. and J.F.L. supervised the work. All authors contributed extensively to discussion. P.D. wrote the paper with the support and correction of all other authors. The authors declare no competing interests.

6.6 Supplementary Material

6.6.1 Hardness evolution

In Fig. 6.5 the hardness evolution of the alloy 6016 for natural aging is given.

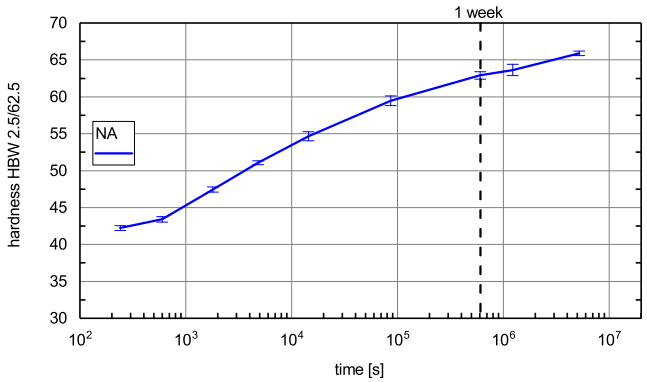

Figure 6.5: Hardness evolution over natural aging time, after solution heat treatment and quenching. An additional line for one week is added as visual guideline.

6.6.2 Pair Correlation and Radial Distribution Functions

The pair correlation function can be defined as $g(R)_{AA}$ as in Equation 6.2 [138].

$$g\left(R\right)_{AA} = \frac{\mathrm{RDF}\left(R\right)_{AA}}{\rho 4\pi R^2 dR} \tag{6.2}$$

$\mathrm{RDF}\left(R\right)_{AA}$ is the radial distribution function defined for atom probe data (disrcete spatial points) for one atomic species, auto-correlation, the equation enhanced for cross-correlation (AB), as also used in this paper, is given in Equation 6.3. Sometimes the left hand side of Equation 6.2 is called RDF [140], wich can lead to confusion. However, we stick to the defined names as in Equation 6.2 and 6.3. The denominator in Equation 6.2 can itself be seen as the RDF of randomly distributed positions of atoms, in an explicit form for an infinitly expanded medium. In atom probe tomography, comparators for randomly distributed solute atoms are built with radom labeling. This means random sampling without replacement on the existing (x,y,z) coordinates, for the given number of solutes. Therefore Equation 6.2 can be re-written to Equation 6.4. For cross-correlation we define $\mathrm{RDF}\left(R\right)_{AB,\mathrm{rand}}$ as in Equation 6.5, meaning the positions of one species is fixed (B) and the other species (A) is randomly distributed on the remaining possible positions. Pair correlation values (Equation 6.4) > 1 can already indicate clustering of solute atoms, but we use it in a modified way: the ratio of cumulative summed RDFs (Equation 6.7). More than one random drawing is used to characterize the random distribution of solutes

(Equation 6.6). For the calculation of the measure $f(r)$ the mean curve of the $h(r)_i$ random comparators are used (Equation 6.7). The upper boundary for the transformed standard deviation (Std) of $h(r)_i$ is calculated via Equation 6.8 and the lower via Equation 6.9.

$$\mathrm{RDF}\,(R)_{AB} = \sum_{k=1}^{n_A}\left(\sum_{\substack{l=1,\\ l\neq k \text{ if } A=B}}^{n_B}\left(\mathrm{Hist}\left(\left|\left|\vec{P}_{A,k}-\vec{P}_{B,l}\right|\right|_2\right)\right)\right) \tag{6.3}$$

$$g\,(R)_{AA} = \frac{\mathrm{RDF}\,(R)_{AA}}{\mathrm{RDF}\,(R)_{AA,\mathrm{rand}}} \tag{6.4}$$

$$\mathrm{RDF}\,(R)_{AB,\mathrm{rand}} = \sum_{k=1}^{n_A}\left(\sum_{\substack{l=1,\\ l\neq k \text{ if } A=B}}^{n_B}\left(\mathrm{Hist}\left(\left|\left|\vec{P}_{A_{\mathrm{rand}},k}-\vec{P}_{B,l}\right|\right|_2\right)\right)\right) \tag{6.5}$$

$$h(r)_i = \sum_{R=0}^{r}\left(\mathrm{RDF}\,(R)_{AB,\mathrm{rand},i}\right) \tag{6.6}$$

$$f(r) = \frac{\sum_{R=0}^{r}\left(\mathrm{RDF}\,(R)_{AB}\right)}{\mathrm{Mean}\left(h\,(r)_i\right)},\ i = 1...40 \tag{6.7}$$

$$\mathrm{err\_up}\,(f(r)) = \frac{\sum_{R=0}^{r}\left(\mathrm{RDF}\,(R)_{AB}\right)}{\mathrm{Mean}\,(h(r)_i) - \mathrm{Std}\,(h(r)_i)} \tag{6.8}$$

$$\mathrm{err\_low}\,(f(r)) = \frac{\sum_{R=0}^{r}\left(\mathrm{RDF}\,(R)_{AB}\right)}{\mathrm{Mean}\,(h(r)_i) + \mathrm{Std}\,(h(r)_i)} \tag{6.9}$$

6.6.3 Si migration/surface relaxation and regions of interests

In Fig. 6.6 low Si-Si ratios for the whole datasets can be seen, and the signals are getting less over the natural aging time for both samples. This is attributed to Si migration/surface relaxation [190] during the APT experiment, caused by prefered retention due to the higher evaporation field. We suggest that the distance which Si can travel stays constant, but the overall sample surface area increases, which in overall decreases the signal from migrating Si atoms with increasing sample radius. The amount of Si migration is especially large if the (111) pole [114] (Fig. 6.8, 6.10) or the zone line (111)-(022) (Fig. 6.9) is visible on the detector hitmap, which explains the general higher Si-Si signal for "nano_tip_aged_02". Artefacts of this kind can be minimized with choosing regions of interests (Fig. 6.7, 6.8, 6.9, 6.10, 6.11), but cannot be ruled out completly (compare Fig. 6.6 a) and b) and also Fig. 6.12 a) and b)).

In Fig. 6.12 the results for the replicant measurements for the whole datasets a) and region of interests b) are shown. The already obtained results do confirm themselves. For Si-Si ratios again the influence of the Si migration can be seen when analyzing the whole dataset.

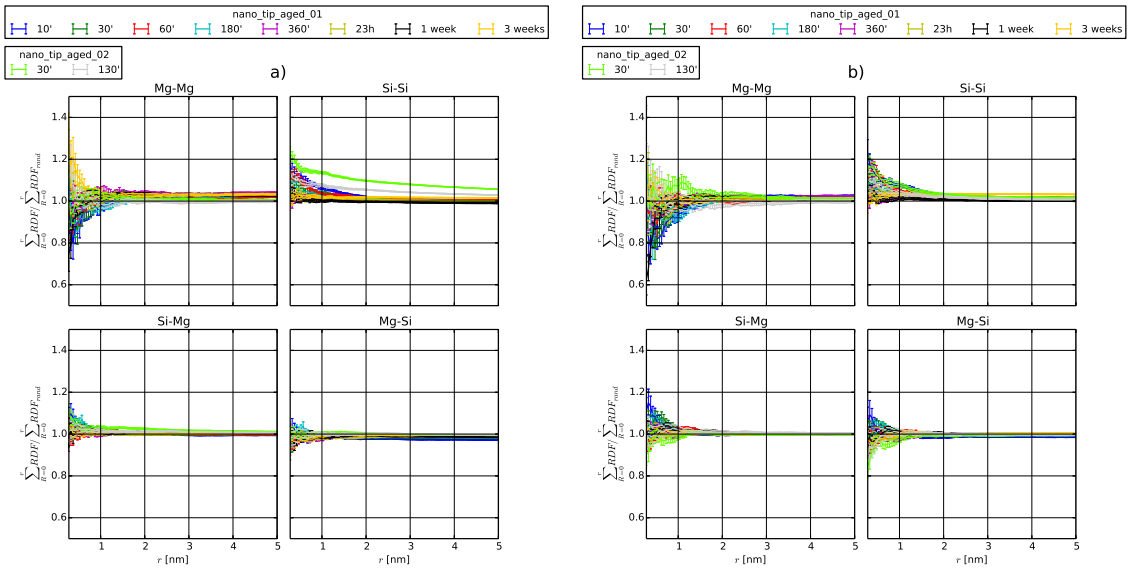

Figure 6.6: In-tip natural aging after solution heat treatment and preparation of samples at -40° C. Analysis of the spatial positions of solute atoms: Shown is the ratio of Equation 6.1 for the given interactions (Mg-Mg, Si-Si, Si-Mg and Mg-Si) for "nano_tip_aged" samples, see Table 6.1. Values > 1 for the ratio indicate clustering. Results shown in a) correspond to the whole dataset, in comparison to b) (same data as in main) where regions of interest are shown.

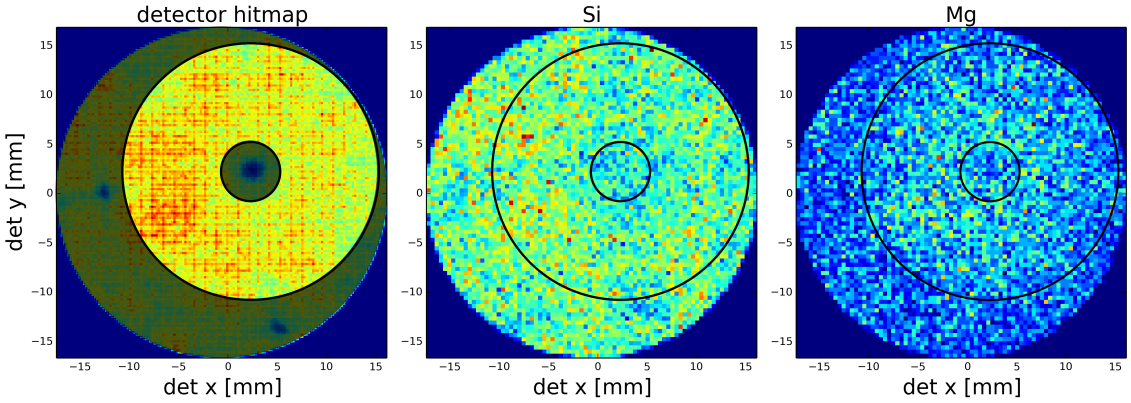

Figure 6.7: "nano_tip_aged_01" detector hitmaps for all atoms "detector hitmap", Si atoms and Mg atoms respectively. Regions of interest: Neglected data is darkened in the detector hitmap and the borderline used is drawn. The used borderlines are also drawn in the Si and Mg detector hitmaps. The region of interest is centered around the (002) pole.

Table 6.1: Overview of examined experiments and samples.

Sample name	Time bulk RT	Prep. T	tip-aging t at RT	Mg[%]	Si[%]	Size
nano_tip_aged_01	0	-40° C	10’	0.27	0.97	5.2
			30’	0.27	0.96	5.6
			60’	0.27	0.95	5.6
			180’	0.26	0.94	6.3
			360’	0.26	0.94	6.3
			23h	0.26	0.93	7.6
			1 week	0.26	0.95	8.9
			3 weeks	0.25	0.92	11.5
nano_tip_aged_02	0	-40° C	30’	0.27	1.05	9
			130’	0.26	1.00	6.9
bulk_aged_01	9’	RT	3 weeks	0.36	1.02	21.4
			6 weeks	0.35	1.01	10.2
bulk_aged_02	1 week	RT	2 weeks	0.40	1.20	6.3
bulk_aged_03	1 week	RT	1 day	0.38	0.97	2.8

[1] Time at room temperature after quenching at bulk dimensions (rods 0.7×0.7×20 mm).
[2] Ambient temperature of the sample preparation location.
[3] Time during natural aging in the tip dimensions.
[4] Mg, molar percent of the used reconstruction, no un-ranged ions counted.
[5] Si, molar percent of the used reconstruction, no un-ranged ions counted.
[6] Number of atoms, in millions, for the used .pos file (whole dataset).

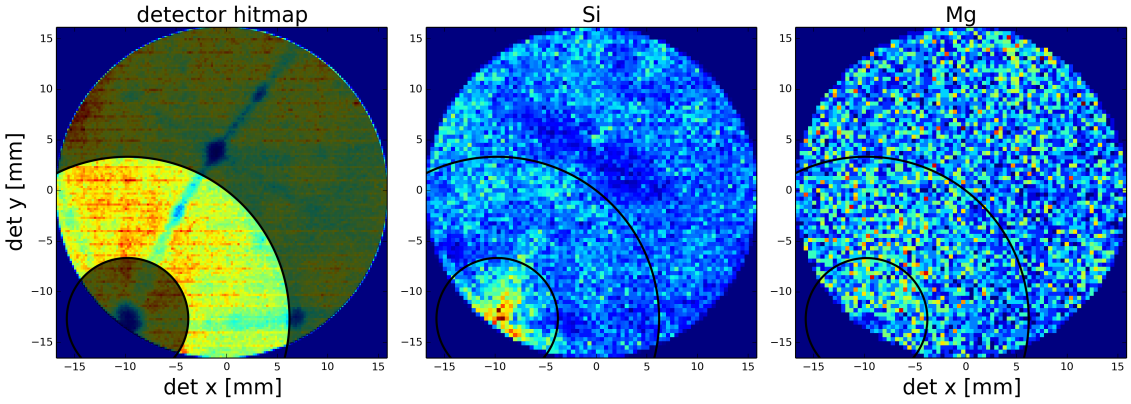

Figure 6.8: ”nano_tip_aged_02” detector hitmaps for all atoms ”detector hitmap”, Si atoms and Mg atoms respectively. Regions of interest: Neglected data is darkened in the detector hitmap and the borderline used is drawn. The used borderlines are also drawn in the Si and Mg detector hitmaps. The region of interest is centered around the (111) pole.

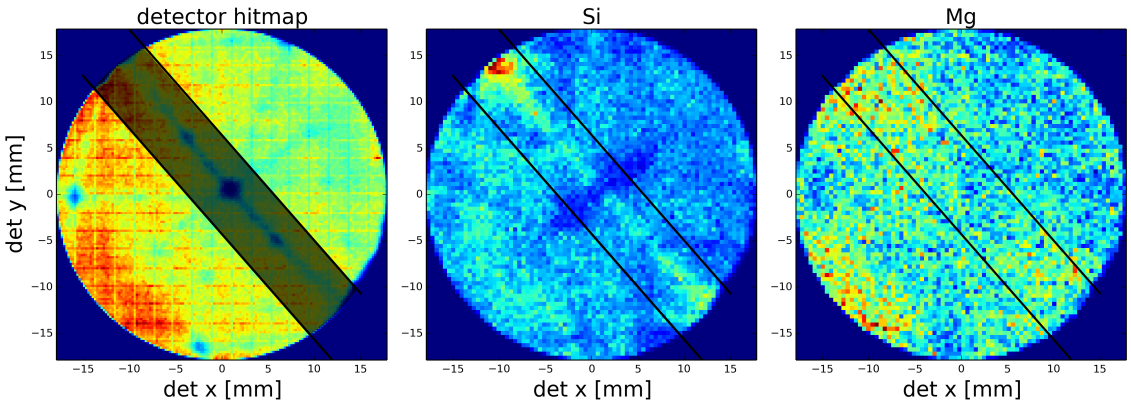

Figure 6.9: "bulk_aged_01" detector hitmaps for all atoms "detector hitmap", Si atoms and Mg atoms respectively. Regions of interest: Neglected data is darkened in the detector hitmap and the borderline used is drawn. The used borderlines are also drawn in the Si and Mg detector hitmaps. The neglected data is near the (111)-(022) zone line, the region of interest is the supplementary area.

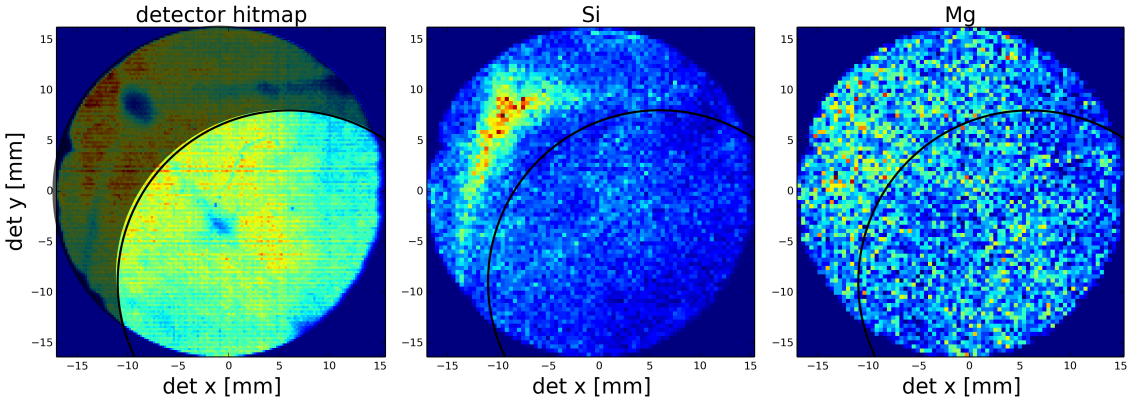

Figure 6.10: "bulk_aged_02" detector hitmaps for all atoms "detector hitmap", Si atoms and Mg atoms respectively. Regions of interest: Neglected data is darkened in the detector hitmap and the borderline used is drawn. The used borderlines are also drawn in the Si and Mg detector hitmaps. The (002) pole is seen at the edge of the detector hitmap in the fourth quadrant.

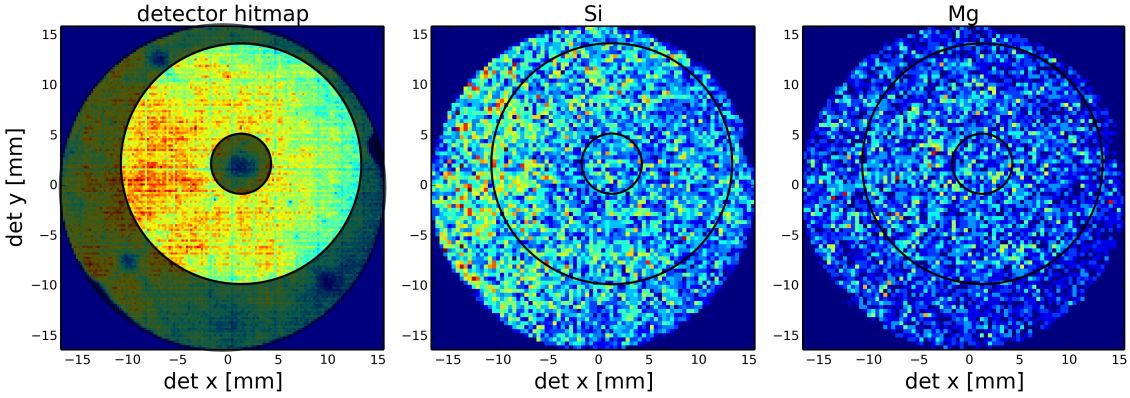

Figure 6.11: "bulk_aged_03" detector hitmaps for all atoms "detector hitmap", Si atoms and Mg atoms respectively. Regions of interest: Neglected data is darkened in the detector hitmap and the borderline used is drawn. The used borderlines are also drawn in the Si and Mg detector hitmaps. The region of interest is centered around the (002) pole.

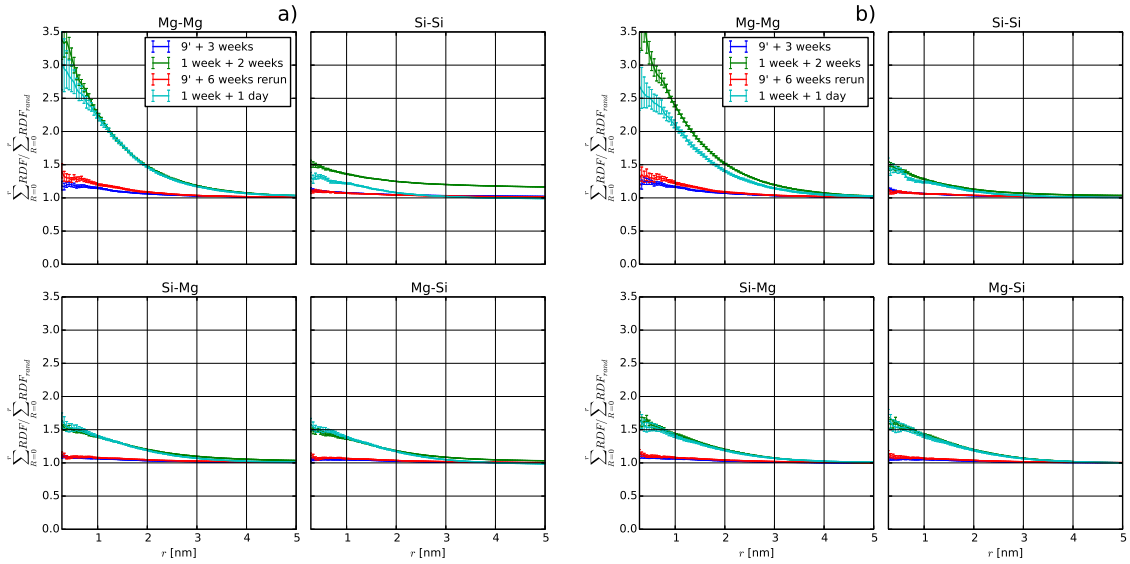

Figure 6.12: Bulk-aging after solution heat treatment and further in-tip aging, shown are the ratio of Equation 6.1 for the given interactions (Mg-Mg, Si-Si, Mg-Si, Si-Mg). Values > 1 for the ratio indicate clustering. Blue lines correspond to the sample "bulk_aged_01", green lines correspond to the sample "bulk_aged_02" as in Fig. 6.4. Additionally in red is the rerun of (9' + 3 weeks) after further 3 weeks of in-tip natural aging (9' + 6 weeks). A reference sample, "bulk_aged_03", is shown in cyan, with a bulk aging time of 1 week and a further in-tip aging time of 1 day (1 week + 1 day). Shown are the results for the whole dataset in a) and the results for the region of interests in b). The already obtained results do confirm themselves: The (1 week + 1 day) shows ratios corresponding to (1 week + 2 weeks) and the rerun of (9' + 3 weeks), (9'+ 6 weeks), shows again for in-tip aging almost no change.

Table 6.2: Cluster search parameters for the used "create cluster analysis" method with IVAS 3.6.12 and resulting number densities. References are given from which the used cluster search parameters are gained.

$d_{\max}$	Order	$N_{\min}$	L	d_{erode}	Nr. density $\times 10^{22}$ $1/m^3$	Ref.	ref. Nr. density$\times 10^{22}$ $1/m^3$
0.700	1	10	0.700	0.00	58	[129, 130, 133]	122
0.750	1	10	0.750	0.00	108	[131]	130

[1] 1 week NA, LAR3DAP ($\sim$ detection efficiency as reflectron fitted LEAP), 0.51%Mg, 0.94%Si atomic, maximum separation cluster search [118], $(d_{\max}, N_{\min})$, Data in Brief Table 6

[2] 300 h (12 days) NA , LEAP 3000 HR [40], 0.62%Mg, 0.93%Si mass, maximum separation cluster search [192], $(d_{\max}, N_{\min})$, datapoint from Fig. 6

6.6.4 Clustersearch

For comparison to literature data, additionally a cluster search with the commercial program IVAS 3.6.12 ("create cluster analysis") was carried out for "bulk_aged_02". The differently used cluster search parameters and resulting number densities are given in Table 6.2. The number density is not a very robust measure and can strongly depent on input parameters as can be seen. However, our measurements do correspond to the values of the literature for 1 week NA. Although several limitations should be considered: maximum seperation cluster search could depent on which software version is used for analysis [62]. Different compositions are used in different literature studies. It is visually obtained that parameters as in Ref. [129, 130, 133] for this dataset lead to an underestimation of the amount of clusters, possibly due to difference in solute content.

6.6.5 Non-equilibrium vacancy evolution

For completeness additionally in Fig. 6.13 the vacancy evolution is simulated for the used alloy with vacancy trapping for Mg and Si in comparison to the results of main Fig. 6.3.

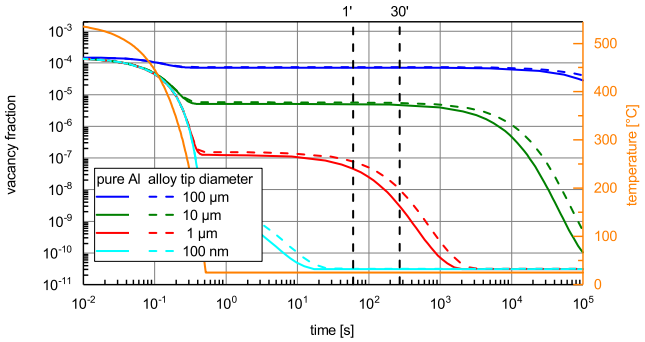

Figure 6.13: Simulation of the non-equilibrium vacancy evolution. Calculated non-equilibrium vacancy fraction over time and temperature upon quenching and natural aging for pure Al and the used alloy (dashed lines) with solutes as trapping sites. Vacancy binding energies of 5000, 1000 and 0 J/mol for respectively Si, Mg and Cu are used. [193] Additional lines for 1' and 30' are added as visual guidelines.

Chapter 7

Summary & Outlook

An extensive literature survey on existing information of clustering in aluminum alloys was realized within a review paper included in the introduction of the thesis. Characterization methods were discussed in general and results for the important AlMgSi alloys were presented in detail. Results of indirect methods and especially, results of atom probe tomography, as direct method, were investigated and summarized.

In-depth understanding of existing analysis methods in atom probe tomography was established and an independent data analysis approach, on a script basis, was formulated. Starting from considerations regarding the reconstruction algorithm, over the ranging process, to the spatial analysis as major part, several methods were implemented. Thereof a parameter free analysis method, based on the radial distribution function formalism, is chosen to be used as best measure for clustering.

A special sample production strategy was developed and applied to access the time-region below 60-100 min of natural aging via atom probe tomography: including quenching samples into LN_2, micro-polishing at arctic temperatures and a cryo-transfer to an atom probe.

With the first results of the realized experiments it could be shown that, no significant cluster amount can be detected after quenching AlMgSi alloys, and the as-quenched state can be accessed, with atom probe tomography.

Moreover, experiments with *in-situ* natural aging of atom probe tomography tips did not show an increasing amount of the clustering measure with increasing time. Samples natural aged at bulk material in comparison showed significant amount of clustering. It is concluded that natural aging not only depends on the storage time at room temperature, as is known in natural aging for more than 110 years, but also shows a pronounced size effect.

It was reasoned that the sample surface acts as vacancy sink. With simulations it was demonstrated clearly that non-equilibrium diffusion effectively stops very quickly, when the sample size approaches the nanometer range. Moreover it was shown that, for small dimensions it is also impossible to reach a significant fraction of non-equilibrium vacancies upon rapid quenching.

This findings not only affect clustering in Al alloys, but all substitutional diffusion processes in metals based on non-equilibrium vacancies.
In general the size dependency has to be considered when *in-situ* high resolution microscopy techniques, such as transmission electron microscopy or atom probe tomography, are utilised to study non-equilibrium kinetics in bulk materials. We conclude that atom probe tomography results for clustering over the last 20 years may have been influenced by ill definition of the applied natural aging time, this could explain differing results on short natural aging. With this thesis a basis was created for the critical assessment of published literature regarding clustering in aluminum alloys. Based on our investigations of the relationship of hitmap to region-of-interest choice, future APT investigations will be able to estimate the effect of field-evaporation artefacts on the spatial analysis for clustering. Further, additional experiments with the applied approach will give consistent information on the evolution of clustering in AlMgSi and other Al alloys. A wrong linkage between kinetics of non-equilibrium vacancy driven reactions at bulk dimensions and the kinetics at nanometer dimensions can now be avoided.

"God made the bulk; surfaces were invented by the devil." – Wolfgang Pauli

Bibliography

[1] F. Ostermann, *Anwendungstechnologie Aluminium*, Vol. 3 (Springer Vieweg, Berlin, Heidelberg, 2015).

[2] P. Brenner and H. Kostron, Zeitschrift für Metallkunde **31. Jahrgang**, 89 (1939).

[3] P. Dumitraschkewitz, S. S. A. Gerstl, L. T. Stephenson, P. J. Uggowitzer, and S. Pogatscher, Advanced Engineering Materials **2012**, 1800255 (2018).

[4] J. Banhart, C. S. T. Chang, Z. Liang, N. Wanderka, M. D. H. Lay, and A. J. Hill, Advanced engineering materials **12**, 559 (2010).

[5] Gesamtverband der Aluminiumindustrie (GDA) e.V., (2012).

[6] M. Murayama and K. Hono, Acta Materialia **47**, 1537 (1999).

[7] G. A. Edwards, K. Stiller, G. L. Dunlop, and M. J. Couper, Acta Materialia **46**, 3893 (1998).

[8] Simon P. Ringer, Kazuhiro Hono, Toshio Sakurai, and Ian J. Polmear, Scripta Materialia **36** (1997).

[9] S. Kim, J. Kim, H. Tezuka, E. Kobayashi, and T. Sato, Materials Transactions **54**, 297 (2013).

[10] Y. Birol, Materials Science and Engineering: A **391**, 175 (2005).

[11] C. S. T. Chang, I. Wieler, N. Wanderka, and J. Banhart, Ultramicroscopy **109**, 585 (2009).

[12] S. Pogatscher, H. Antrekowitsch, H. Leitner, T. Ebner, and P. J. Uggowitzer, Acta Materialia **59**, 3352 (2011).

[13] R. Marceau, G. Sha, R. Ferragut, A. Dupasquier, and S. P. Ringer, Acta Materialia **58**, 4923 (2010).

[14] A. Wilm, Metallurgie: Zeitschrift für die gesamte Hüttenkunde **8**, 225 (1911).

[15] N. E. Prasad and R. J. Wanhill, *Aerospace Materials and Material Technologies* (Springer, 2017).

[16] S. P. Ringer, Materials Science Forum **519-521**, 25 (2006).

[17] C. D. Marioara, H. Nordmark, S. J. Andersen, and R. Holmestad, Journal of Materials Science **41**, 471 (2006).

[18] E. A. Mørtsell, C. D. Marioara, S. J. Andersen, J. Røyset, O. Reiso, and R. Holmestad, Metallurgical and Materials Transactions A **46**, 4369 (2015).

[19] C. D. Marioara, S. J. Andersen, J. Jansen, and H. W. Zandbergen, Acta Materialia **49**, 321 (2001).

[20] P. M. Derlet, S. J. Andersen, C. D. Marioara, and A. Frøseth, Journal of Physics: Condensed Matter **14**, 4011 (2002).

[21] S. J. Andersen, H. W. Zandbergen, J. Jansen, C. Traeholt, U. Tundal, and O. Reiso, Acta Materialia **46**, 3283 (1998).

[22] R. Vissers, M. A. van Huis, J. Jansen, H. W. Zandbergen, C. D. Marioara, and S. J. Andersen, Acta Materialia **55**, 3815 (2007).

[23] S. J. Andersen, C. D. Marioara, R. Vissers, A. Frøseth, and H. W. Zandbergen, Materials Science and Engineering: A **444**, 157 (2007).

[24] S. J. Andersen, C. D. Marioara, A. Frøseth, R. Vissers, and H. W. Zandbergen, Materials Science and Engineering: A **390**, 127 (2005).

[25] R. Vissers, C. D. Marioara, S. J. Andersen, and R. Holmestad, Aluminium Alloys. Their Physical and Mechanical Properties , 1263 (2008).

[26] J. Banhart, ASM Handbook **4E** (2016).

[27] M. H. Jacobs, The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics **26**, 1 (1972).

[28] S. D. Dumolt, D. E. Laughlin, and J. C. Williams, Scripta Metallurgica **18**, 1347 (1984).

[29] D. Chakrabarti and D. E. Laughlin, Progress in Materials Science **49**, 389 (2004).

[30] C. D. Marioara, S. J. Andersen, T. N. Stene, H. Hasting, J. Walmsley, Van Helvoort, A. T. J., and R. Holmestad, Philosophical Magazine **87**, 3385 (2007).

[31] J. M. Silcock, Journal of the Institute of Metals **89**, 203 (1961).

[32] A. Deschamps, T. J. Bastow, F. de Geuser, A. J. Hill, and C. R. Hutchinson, Acta Materialia **59**, 2918 (2011).

[33] S. C. Wang, M. J. Starink, and N. Gao, Scripta Materialia **54**, 287 (2006).

[34] S. C. Wang and M. J. Starink, Materials Science and Engineering: A **386**, 156 (2004).

[35] G. Bergman, J. Waugh, and L. Pauling, Acta Cryst. **10**, 254 (1957).

[36] J. H. Chen, E. Costan, M. A. van Huis, Q. Xu, and H. W. Zandbergen, Science (New York, N.Y.) **312**, 416 (2006).

[37] S. J. Andersen, C. D. Marioara, J. Friis, R. Bjørge, Q. Du, I. G. Ringdalen, S. Wenner, E. A. Mørtsell, R. Holmestad, T. Saito, J. Røyset, and O. Reiso, in *Materials Science Forum*, Vol. 877 (2017) pp. 461–470.

[38] S. Pogatscher, H. Antrekowitsch, H. Leitner, D. Pöschmann, Z. L. Zhang, and P. J. Uggowitzer, Acta Materialia **60**, 4496 (2012).

[39] S. Pogatscher, H. Antrekowitsch, T. Ebner, and P. Uggowitzer, Light Metals 2012 , 415 (2012).

[40] Y. Aruga, M. Kozuka, Y. Takaki, and T. Sato, Metallurgical and Materials Transactions A **45**, 5906 (2014).

[41] S. Pogatscher, E. Kozeschnik, H. Antrekowitsch, M. Werinos, S. Gerstl, J. F. Löffler, and P. J. Uggowitzer, Scripta Materialia **89**, 53 (2014).

[42] F. D. Fischer, J. Svoboda, F. Appel, and E. Kozeschnik, Acta Materialia **59**, 3463 (2011).

[43] K. Takata, K. Ushioda, R. Akiyoshi, K.-i. Ikeda, J. Takahashi, S. Hata, and K. Kaneko, Materials Transactions **58**, 728 (2017).

[44] S. Pogatscher, H. Antrekowitsch, M. Werinos, F. Moszner, S. S. A. Gerstl, M. F. Francis, W. A. Curtin, J. F. Löffler, and P. J. Uggowitzer, Physical Review Letters **112**, 225701 (2014).

[45] M. Werinos, H. Antrekowitsch, T. Ebner, R. Prillhofer, P. J. Uggowitzer, and S. Pogatscher, Materials & Design **107**, 257 (2016).

[46] S. Pogatscher, H. Antrekowitsch, M. Werinos, G. Rank, A. Kaiß, R. Prillhofer, Löffler, Jörg, F., and P. Uggowitzer, Light Metals 2015 , 265 (2015).

[47] M. Werinos, H. Antrekowitsch, E. Kozeschnik, T. Ebner, F. Moszner, J. F. Löffler, P. J. Uggowitzer, and S. Pogatscher, Scripta Materialia **112**, 148 (2016).

[48] M. Liu and J. Banhart, Materials Science and Engineering: A **658**, 238 (2016).

[49] M. Werinos, H. Antrekowitsch, T. Ebner, R. Prillhofer, W. A. Curtin, P. J. Uggowitzer, and S. Pogatscher, Acta Materialia **118**, 296 (2016).

[50] H. K. Hardy, Journal of the Institute of Metals **78**, 169 (1950).

[51] C. Wolverton, Acta Materialia **55**, 5867 (2007).

[52] R. Marceau, G. Sha, R. N. Lumley, and S. P. Ringer, Acta Materialia **58**, 1795 (2010).

[53] K. Strobel, M. D. Lay, M. A. Easton, L. Sweet, S. Zhu, N. C. Parson, and A. J. Hill, Materials Characterization **111**, 43 (2016).

[54] G. Gottstein, *Physikalische Grundlagen der Materialkunde*, 3rd ed., Springer-Lehrbuch (Springer, Berlin, Heidelberg, 2007).

[55] F. R. Fickett, Cryogenics **11**, 349 (1971).

[56] Stefan Pogatscher, *Phase transitions in quenched nonferrous metallic systems*, Habilitationsschrift, Montanuniversität Leoben, Leoben (2017).

[57] C. Panseri and T. Federighi, J. Inst. Metals **94**, 99 (1966).

[58] A. J. Hillel and P. L. Rossiter, Philosophical Magazine B **44**, 383 (1981).

[59] I. Kovacs, J. Lendvai, and E. Nagy, Acta Metallurgica **20**, 975 (1972).

[60] Shoichi Hirosawa, Tatsuo Sato, Junichi Yokota, and Akihiko Kamio, Materials Transactions, JIM **39**, 139 (1998).

[61] H. S. Zurob and H. Seyedrezai, Scripta Materialia **61**, 141 (2009).

[62] L. Cao, P. A. Rometsch, and M. J. Couper, Materials Science and Engineering: A **559**, 257 (2013).

[63] H. Seyedrezai, D. Grebennikov, P. Mascher, and H. S. Zurob, Materials Science and Engineering: A **525**, 186 (2009).

[64] S. Esmaeili, D. Vaumousse, M. W. Zandbergen, W. J. Poole, A. Cerezo, and D. J. Lloyd, Philosophical Magazine **87**, 3797 (2007).

[65] Mittemeijer, Journal of Materials Science **27**, 3977 (1992).

[66] B. Milkereit, O. Kessler, and C. Schick, Thermochimica Acta **492**, 73 (2009).

[67] B. Milkereit, N. Wanderka, C. Schick, and O. Kessler, Materials Science and Engineering: A **550**, 87 (2012).

[68] M. Luckabauer, E. Hengge, G. Klinser, W. Sprengel, and R. Würschum, in *Magnesium Technology 2017*, edited by K. N. Solanki, D. Orlov, A. Singh, and N. R. Neelameggham (Springer International Publishing, Cham, 2017) pp. 669–674.

[69] J. Osten, B. Milkereit, C. Schick, and O. Kessler, Materials **8**, 2830 (2015).

[70] I. Dutta and S. M. Allen, Journal of Materials Science Letters **10**, 323 (1991).

[71] A. K. Gupta, D. J. Lloyd, and S. A. Court, Materials Science and Engineering: A **301**, 140 (2001).

[72] A. K. Gupta, D. J. Lloyd, and S. A. Court, Materials Science and Engineering: A **316**, 11 (2001).

[73] A. Serizawa, S. Hirosawa, and T. Sato, Metallurgical and Materials Transactions A **39**, 243 (2008).

[74] S. Esmaeili, D. J. Lloyd, and W. J. Poole, Acta Materialia **51**, 3467 (2003).

[75] C. S. T. Chang and J. Banhart, Metallurgical and Materials Transactions A **42**, 1960 (2011).

[76] Y. Birol and M. Karlik, Scripta Materialia **55**, 625 (2006).

[77] Y. Yan, *Investigation of the negative and positive effects of natural aging on artificial aging response in Al-Mg-Si alloys*, Phd. thesis, Technischen Universität Berlin, Berlin (2014).

[78] H. E. Kissinger, Analytical chemistry **29**, 1702 (1957).

[79] S. Pogatscher, H. Antrekowitsch, and P. J. Uggowitzer, Materials Letters **100**, 163 (2013).

[80] S. Pogatscher, H. Antrekowitsch, and P. J. Uggowitzer, Acta Materialia **60**, 5545 (2012).

[81] O. R. Mhyr, O. Grong, and S. J. Andersen, Acta Materialia **49** (2001).

[82] M. J. Starink, L. F. Cao, and P. A. Rometsch, Acta Materialia **60**, 4194 (2012).

[83] M. J. Starink and S. C. Wang, Acta Materialia **57**, 2376 (2009).

[84] Q. Zhao, Scripta Materialia **84-85**, 43 (2014).

[85] Y. Birol, Scripta Materialia **54**, 2003 (2006).

[86] H. Zhong, P. A. Rometsch, Q. Zhu, L. Cao, and Y. Estrin, Materials Science and Engineering: A **687**, 323 (2017).

[87] J. Banhart, M. D. H. Lay, C. S. T. Chang, and A. J. Hill, Physical Review B **83**, 89 (2011).

[88] M. Liu, J. Čížek, C. S. Chang, and J. Banhart, Acta Materialia **91**, 355 (2015).

[89] P. Schloth, A. Menzel, J. L. Fife, J. N. Wagner, H. van Swygenhoven, and J.-M. Drezet, Scripta Materialia **108**, 56 (2015).

[90] P. Schloth, J. N. Wagner, J. L. Fife, A. Menzel, and J.-M. Drezet, Applied Physics Letters **105** (2014).

[91] K. Nishimura, K. Matsuda, Q. Lei, T. Namiki, S. Lee, N. Nunomra, T. Matsuzaki, and W. D. Hutchison, Materials Transactions (2016).

[92] S. Wenner, R. Holmestad, K. Matsuda, K. Nishimura, T. Matsuzaki, D. Tomono, F. L. Pratt, and C. D. Marioara, Physical Review B **86**, 126 (2012).

[93] S. Wenner, K. Nishimura, K. Matsuda, T. Matsuzaki, D. Tomono, F. L. Pratt, C. D. Marioara, and R. Holmestad, Acta Materialia **61**, 6082 (2013).

[94] M. K. Miller and R. Forbes, *Atom probe tomography: The local electrode atom probe* (Springer, New York, 2014).

[95] B. Gault, M. P. Moody, J. M. Cairney, and S. P. Ringer, eds., *Atom probe microscopy*, Springer Series in Materials Science, Vol. 160 (Springer, New York, 2012).

[96] D. J. Larson, T. J. Prosa, Ulfig, Robert, M., B. P. Geiser, and T. F. Kelly, *Local electrode atom probe tomography: A User's Guide* (Springer, New York, 2013).

[97] J. Buha, R. N. Lumley, A. G. Crosky, and K. Hono, Acta Materialia **55**, 3015 (2007).

[98] M. K. Miller, A. Cerezo, M. G. Hetherington, and G. D. W. Smith, *Atom probe field ion microscopy* (Oxford Science Publications - Clarendon Press, Oxford, 1996).

[99] P. Bas, A. Bostel, B. Deconihout, and D. Blavette, Applied Surface Science **87-88**, 298 (1995).

[100] F. Vurpillot, B. Gault, B. P. Geiser, and D. J. Larson, Ultramicroscopy **132**, 19 (2013).

[101] Y. Aruga, M. Kozuka, and T. Sato, Journal of Alloys and Compounds , 1115 (2018).

[102] L. T. Stephenson, M. P. Moody, B. Gault, and S. P. Ringer, Microscopy research and technique **74**, 799 (2011).

[103] B. Gault, X. Y. Cui, M. P. Moody, F. de Geuser, C. Sigli, S. P. Ringer, and A. Deschamps, Scripta Materialia **66**, 903 (2012).

[104] T. Boll, T. Al-Kassab, Y. Yuan, and Z. G. Liu, Ultramicroscopy **107**, 796 (2007).

[105] M. P. Moody, B. Gault, L. T. Stephenson, R. K. W. Marceau, R. C. Powles, A. V. Ceguerra, A. J. Breen, and S. P. Ringer, Microscopy and microanalysis **17**, 226 (2011).

[106] M. P. Moody, A. V. Ceguerra, A. J. Breen, X. Y. Cui, B. Gault, L. T. Stephenson, R. K. W. Marceau, R. C. Powles, and S. P. Ringer, Nature communications **5**, 5501 (2014).

[107] B. P. Geiser, T. F. Kelly, D. J. Larson, J. Schneir, and J. P. Roberts, Microscopy and microanalysis **13**, 437 (2007).

[108] B. P. Geiser, D. J. Larson, E. Oltman, S. Gerstl, D. Reinhard, T. F. Kelly, and T. J. Prosa, Microscopy and Microanalysis **15**, 292 (2009).

[109] B. Gault, M. P. Moody, F. de Geuser, G. Tsafnat, A. La Fontaine, L. T. Stephenson, D. Haley, and S. P. Ringer, Journal of Applied Physics **105**, 034913 (2009).

[110] S. K. Suram and K. Rajan, Ultramicroscopy **132**, 136 (2013).

[111] F. de Geuser and B. Gault, Microscopy and Microanalysis **23**, 238 (2017).

[112] Daniel Beinke, Christian Oberdorfer, and Guido Schmitz, Ultramicroscopy **165**, 34 (2016).

[113] B. Gault, F. Danoix, K. Hoummada, D. Mangelinck, and H. Leitner, Ultramicroscopy **113**, 182 (2012).

[114] S. Pogatscher, S. Gerstl, J. F. Löffler, and P. J. Uggowitzer, Acta Physica Polonica A **128**, 643 (2015).

[115] E. A. Marquis and F. Vurpillot, Microscopy and microanalysis **14**, 561 (2008).

[116] J. M. Hyde and C. A. English, MRS Proceedings **650**, R6.6 (2000).

[117] L. Ertöz, M. Steinbach, and V. Kumar, in *Proceedings of the 2003 SIAM International Conference on Data Mining* (2003) pp. 47–58.

[118] D. Vaumousse, A. Cerezo, and P. J. Warren, Ultramicroscopy **95**, 215 (2003).

[119] L. T. Stephenson, M. P. Moody, P. V. Liddicoat, and S. P. Ringer, Microscopy and microanalysis **13**, 448 (2007).

[120] A. V. Ceguerra, M. P. Moody, L. T. Stephenson, R. K. Marceau, and S. P. Ringer, Philosophical Magazine **90**, 1657 (2010).

[121] S. Samudrala, O. Wodo, S. K. Suram, S. Broderick, K. Rajan, and B. Ganapathysubramanian, Computational Materials Science **77**, 335 (2013).

[122] J. Zelenty, A. Dahl, J. Hyde, G. D. W. Smith, and M. P. Moody, Microscopy and Microanalysis **23**, 269 (2017).

[123] W. Lefebvre, T. Philippe, and F. Vurpillot, Ultramicroscopy **111**, 200 (2011).

[124] P. Felfer, A. V. Ceguerra, S. P. Ringer, and J. M. Cairney, Ultramicroscopy **150**, 30 (2015).

[125] S. Srinivasan, K. Kaluskar, S. Dumpala, S. Broderick, and K. Rajan, Ultramicroscopy **159 Pt 2**, 381 (2015).

[126] J. M. Hyde, E. A. Marquis, K. B. Wilford, and T. J. Williams, Ultramicroscopy **111**, 440 (2011).

[127] C. A. Williams, D. Haley, E. A. Marquis, G. D. W. Smith, and M. P. Moody, Ultramicroscopy **132**, 271 (2013).

[128] E. A. Jägle, P.-P. Choi, and D. Raabe, Microscopy and Microanalysis **20**, 1662 (2014).

[129] M. W. Zandbergen, Q. Xu, A. Cerezo, and G. Smith, Acta Materialia **101**, 136 (2015).

[130] M. W. Zandbergen, Q. Xu, A. Cerezo, and G. Smith, Data in Brief **5**, 626 (2015).

[131] Y. Aruga, M. Kozuka, Y. Takaki, and T. Sato, Materials Science and Engineering: A **631**, 86 (2015).

[132] Z. Jia, L. Ding, L. Cao, R. Sanders, S. Li, and Q. Liu, Metallurgical and Materials Transactions A **48**, 459 (2017).

[133] M. W. Zandbergen, A. Cerezo, and G. Smith, Acta Materialia **101**, 149 (2015).

[134] Y. Aruga, M. Kozuka, Y. Takaki, and T. Sato, Scripta Materialia **116**, 82 (2016).

[135] R. Marceau, A. de Vaucorbeil, G. Sha, S. P. Ringer, and W. J. Poole, Acta Materialia **61**, 7285 (2013).

[136] M. Torsæter, H. S. Hasting, W. Lefebvre, C. D. Marioara, J. C. Walmsley, S. J. Andersen, and R. Holmestad, Journal of applied physics **108**, 073527 (2010).

[137] L. Cao, P. A. Rometsch, and M. J. Couper, Materials Science and Engineering: A **571**, 77 (2013).

[138] F. de Geuser, W. Lefebvre, and D. Blavette, Philosophical Magazine Letters **86**, 227 (2006).

[139] T. Philippe, S. Duguay, and D. Blavette, Ultramicroscopy **110**, 862 (2010).

[140] D. Haley, T. Petersen, G. Barton, and S. P. Ringer, Philosophical Magazine **89**, 925 (2009).

[141] A. V. Ceguerra, M. P. Moody, R. C. Powles, T. C. Petersen, R. K. W. Marceau, and S. P. Ringer, Acta crystallographica. Section A, Foundations of crystallography **68**, 547 (2012).

[142] A. Serizawa, T. Sato, and W. J. Poole, Philosophical Magazine Letters **90**, 279 (2010).

[143] S. Pogatscher, H. Antrekowitsch, H. Leitner, A. S. Sologubenko, and P. J. Uggowitzer, Scripta Materialia **68**, 158 (2013).

[144] P. Dumitraschkewitz, S. S. A. Gerstl, P. J. Uggowitzer, J. F. Löffler, and S. Pogatscher, Advanced Engineering Materials **19**, 1600668 (2017).

[145] A. Poznak, R. Marceau, and P. G. Sanders, Materials Science and Engineering: A **721**, 47 (2018).

[146] Y. Aruga, S. Kim, M. Kozuka, E. Kobayashi, and T. Sato, Materials Science and Engineering: A **718**, 371 (2018).

[147] E. A. Marquis and J. M. Hyde, Materials Science and Engineering: R: Reports **69**, 37 (2010).

[148] A. Mottura, N. Warnken, M. K. Miller, M. W. Finnis, and R. C. Reed, Acta Materialia **58**, 931 (2010).

[149] S. S. Gerstl and R. Wepf, Microscopy and Microanalysis **21**, 517 (2015).

[150] D. E. Perea, S. S. A. Gerstl, J. Chin, B. Hirschi, and J. E. Evans, Advanced structural and chemical imaging **3**, 12 (2017).

[151] P. A. Rometsch and L. Cao, in *Proceedings of the 12th International Conference on Aluminium Alloys* (2010) pp. 389–394.

[152] G. H. Tao, C. H. Liu, J. H. Chen, Y. X. Lai, P. P. Ma, and L. M. Liu, Materials Science and Engineering: A **642**, 241 (2015).

[153] B. L. Dorr, Electrical design news **51**, 77 (2006).

[154] M. McCauley, https://github.com/waspinator/AccelStepper (2018).

[155] M. Margolis, *Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects* (O'Reilly Media, Inc., 2012).

[156] A. Knörig, R. Wettach, and J. Cohen, in *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction* (ACM, 2009) pp. 351–358.

[157] G. Sha and S. Ringer, Ultramicroscopy **109**, 580 (2009).

[158] IVAS, *IVAS User Guide 3.6.8* (Cameca Instruments Inc., 2014).

[159] D. Haley, P. Choi, and D. Raabe, Ultramicroscopy **159**, 338 (2015).

[160] A. J. London, D. Haley, and M. P. Moody, Microscopy and Microanalysis **23**, 300 (2017).

[161] D. Saxey, Ultramicroscopy **111**, 473 (2011).

[162] B. Gault, D. Haley, F. de Geuser, M. P. Moody, E. A. Marquis, D. J. Larson, and B. P. Geiser, Ultramicroscopy **111**, 448 (2011).

[163] B. Gault, F. de Geuser, L. T. Stephenson, M. P. Moody, B. C. Muddle, and S. P. Ringer, Microscopy and Microanalysis **14**, 296 (2008).

[164] https://github.com/oscarbranson/apt-tools.

[165] MATLAB, *8.6.0.267246 (R2015b)* (The MathWorks Inc, Natick, Massachusetts, 2015).

[166] M. Muja and D. G. Lowe, IEEE transactions on pattern analysis and machine intelligence **36**, 2227 (2014).

[167] M. P. Moody, B. Gault, L. T. Stephenson, D. Haley, and S. P. Ringer, Ultramicroscopy **109**, 815 (2009).

[168] M. P. Moody, F. Tang, B. Gault, S. P. Ringer, and J. M. Cairney, Ultramicroscopy **111**, 493 (2011).

[169] L. Yao, MethodsX **3**, 268 (2016).

[170] S. Kim, J. Kim, H. Tezuka, E. Kobayashi, and T. Sato, Materials Transactions **54**, 297 (2013).

[171] R. Marceau, R. Ferragut, A. Dupasquier, M. M. Iglesias, and S. P. Ringer, Materials Science Forum **519-521**, 197 (2006).

[172] M. P. Moody, A. Vella, S. S. Gerstl, and P. A. Bagot, MRS Bulletin **41**, 40 (2016).

[173] G. van Rossum and F. L. Drake Jr, *Python reference manual* (Centrum voor Wiskunde en Informatica Amsterdam, 1995).

[174] S. van der Walt, S. C. Colbert, and G. Varoquaux, Computing in Science & Engineering **13**, 22 (2011).

[175] J. D. Hunter *et al.*, Computing in science and engineering **9**, 90 (2007).

[176] Blender Online Community, "Blender 2.71 - a 3d modelling and rendering package," (2014).

[177] L. Campagnola, A. Klein, E. Larson, C. Rossant, and N. P. Rougier, in *Proceedings of the 14th Python in Science Conference* (2015).

[178] J. F. Ziegler, M. D. Ziegler, and J. P. Biersack, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms **268**, 1818 (2010).

[179] S. Rubanov and P. Munroe, Journal of microscopy **214**, 213 (2004).

[180] P. Dumitraschkewitz, S. S. A. Gerstl, P. J. Uggowitzer, J. F. Löffler, and S. Pogatscher, as submitted to Nature communications (April 2019).

[181] F. Appel, D. Herrmann, F. Fischer, J. Svoboda, and E. Kozeschnik, International Journal of Plasticity **42**, 83 (2013).

[182] M. J. Aliaga, R. Schäublin, J. F. Löffler, and M. J. Caturla, Acta Materialia **101**, 22 (2015).

[183] S. Li, Y. Li, Y.-C. Lo, T. Neeraj, R. Srinivasan, X. Ding, J. Sun, L. Qi, P. Gumbsch, and J. Li, International Journal of Plasticity **74**, 175 (2015).

[184] J. Buha, Materials Science and Engineering: A **492**, 11 (2008).

[185] W. Xu, N. Birbilis, G. Sha, Y. Wang, J. E. Daniels, Y. Xiao, and M. Ferry, Nature materials **14**, 1229 (2015).

[186] J. Buha, Materials Science and Engineering: A **489**, 127 (2008).

[187] Z. Mao, C. K. Sudbrack, K. E. Yoon, G. Martin, and D. N. Seidman, Nature materials **6**, 210 (2007).

[188] Y.-S. Chen, D. Haley, S. S. A. Gerstl, A. J. London, F. Sweeney, R. A. Wepf, W. M. Rainforth, P. A. J. Bagot, and M. P. Moody, Science **355**, 1196 (2017).

[189] M. P. Moody, A. Vella, S. S. A. Gerstl, and P. A. J. Bagot, MRS Bulletin **41**, 40 (2016).

[190] C. Oberdorfer, T. Withrow, L.-J. Yu, K. Fisher, E. A. Marquis, and W. Windl, Materials Characterization **146**, 324 (2018).

[191] H. Zhao, B. Gault, D. Ponge, D. Raabe, and F. de Geuser, Scripta Materialia **154**, 106 (2018).

[192] J. M. Hyde and C. A. English, MRS Proceedings **650**, R6.6 (2000).

[193] D. Simonovic and M. H. F. Sluiter, Physical Review B **79**, 054304 (2009).

[194] P. Dumitraschkewitz, H. Clemens, S. Mayer, and D. Holec, Applied Sciences **7**, 1193 (2017).

[195] A.B. Spierings, K. Dawson, P. Dumitraschkewitz, S. Pogatscher, and K. Wegener, Additive Manufacturing **20**, 173 (2018).

Chapter 8

Appendix

8.1 Further publications

Additionally, during the time period of the dissertation the papers (author or co-author): **Impact of Alloying on Stacking Fault Energies in γ-TiAl** [194] and **Microstructure characterization of SLM-processed Al-Mg-Sc-Zr alloy in the heat treated and HIPed condition** [195] were published.

8.2 apt_importers.py

```
"""
several import statements to import libraries
"""
import pandas as pd
#pandas library for special data structures
import struct
#library for handling byte data
import numpy as np
#library specialized on numerical computations
from pyflann import *
#kd-tree libary
import matplotlib.pyplot as plt
#library for plotting
from matplotlib import cm
from scipy.optimize import curve_fit
#curve fitting
from scipy import stats
#statistics
import scipy.linalg
#linear algebra operations
def read_pos(f):
```

```
22      """ Loads an APT .pos file as a pandas dataframe.
23
24      Columns:
25          x: Reconstructed x position
26          y: Reconstructed y position
27          z: Reconstructed z position
28          Da: mass/charge ratio of ion"""
29      # read in the data
30      n = len(file(f).read())/4
31      d = struct.unpack('>'+'f'*n,file(f).read(4*n))
32      # '>' denotes 'big-endian' byte order
33      # unpack data
34      pos = pd.DataFrame({'x': d[0::4],
35                          'y': d[1::4],
36                          'z': d[2::4],
37                          'Da': d[3::4]})
38      return pos
39  def writePos(name,data):
40      """
41      writes binary to name as big-endian float values,
42      is called by writeLpos()
43      """
44      f = open(name,'w')
45      #opens file with write permission, path = name
46      n = len(data)
47      #length of data
48      print([n,type(data)])
49      #output to console
50      d = struct.pack('>'+'f'*n, *data)
51      #packs content of data to byte format (n times big-endian float)
52      f.write(d)
53      #writes data from buffer to disc
54      f.close()
55      #closes file
56  def writeLpos(name, lpos):
57      #saves lpos which is in pandas DataFrame format to the binary .pos
        ↪  file format
58      writePos(name,np.array([lpos.x.get_values(),lpos.y.get_values(),lpos.z.
        ↪  get_values(),lpos.Da.get_values()]).transpose().flatten())
59      #.get_values() ... returns converted pandas file structure to
        ↪  np.array()
60      #.transpose() ... transponses the array
61      #.flatten() ... changes the array to one dimensional array
62  def save_as_epos(name,epos):
63      #saves epos which is in pandas DataFrame format to the binary .epos
        ↪  file format
```

```
64      f = open(name,'w')
65      #opens file with write permission, path = name
66      d = struct.pack('>'+'fffffffffII'*epos.x.size,
        ↪  *epos[['x','y','z','Da','ns','DC_kV','pulse_kV','det_x','det_y','ps
        ↪  lep','ipp']].get_values().flatten())
67      #packs content of *epos to byte format (*epos.x.size times 9 float
        ↪  values followed by two unsigned integers), big-endian
68      f.write(d)
69      #writes data from buffer to disc
70      f.close()
71      #closes file
72  def read_epos(f):
73      """Loads an APT .epos file as a pandas dataframe.
74
75      Columns:
76          x: Reconstructed x position
77          y: Reconstructed y position
78          z: Reconstructed z position
79          Da: Mass/charge ratio of ion
80          ns: Ion Time Of Flight
81          DC_kV: Potential
82          pulse_kV: Size of voltage pulse (voltage pulsing mode only)
83          det_x: Detector x position
84          det_y: Detector y position
85          pslep: Pulses since last event pulse (i.e. ionisation rate)
86          ipp: Ions per pulse (multihits)
87
88       [x,y,z,Da,ns,DC_kV,pulse_kV,det_x,det_y,pslep,ipp].
89          pslep = pulses since last event pulse
90          ipp = ions per pulse
91
92      When more than one ion is recorded for a given pulse, only the
93      first event will have an entry in the "Pulses since last evenT
94      pulse" column. Each subsequent event for that pulse will have
95      an entry of zero because no additional pulser firings occurred
96      before that event was recorded. Likewise, the "Ions Per Pulse"
97      column will contain the total number of recorded ion events for
98      a given pulse. This is normally one, but for a sequence of records
99      a pulse with multiply recorded ions, the first ion record will
100     have the total number of ions measured in that pulse, while the
101     remaining records for that pulse will have 0 for the Ions Per
102     Pulse value.
103         ~ Appendix A of 'Atom Probe tomography: A Users Guide',
104           notes on ePOS format."""
105     # read in the data
106     n = len(file(f).read())/4
```

```
107     #number of numbers in file
108     rs = n / 11
109     #number of rows
110     d = struct.unpack('>'+'fffffffffII'*rs,file(f).read(4*n))
111     # '>' denotes 'big-endian' byte order
112     # unpack data
113     pos = pd.DataFrame({'x': d[0::11],
114                         # beginning with index 0 every 11th element
115                         'y': d[1::11],
116                         'z': d[2::11],
117                         'Da': d[3::11],
118                         'ns': d[4::11],
119                         'DC_kV': d[5::11],
120                         'pulse_kV': d[6::11],
121                         'det_x': d[7::11],
122                         'det_y': d[8::11],
123                         'pslep': d[9::11], # pulses since last event pulse
124                         'ipp': d[10::11]}) # ions per pulse
125     return pos
126     #return DataFrame Object
127 def read_rrng(f):
128     """Loads a .rrng file produced by IVAS. Returns two dataframes of
        ↪  'ions'
129     and 'ranges'."""
130     import re
131     # regular expression library
132
133     rf = open(f,'r').readlines()
134     #file open with 'r' read permission
135     #.readlines() ... creates list of lines
136
137     patterns =
        ↪  re.compile(r'Ion([0-9]+)=([A-Za-z0-9]+).*|Range([0-9]+)=(\d+.\d+)
        ↪  +(\d+.\d+) +Vol:(\d+.\d+) +([A-Za-z:0-9 ]+) +Color:([A-Z0-9]{6})')
138     # regular expression pattern for reading rrange file
139
140     ions = []
141     #create empty list
142     rrngs = []
143     for line in rf:
144         m = patterns.search(line)
145         if m:
146             if m.groups()[0] is not None:
147                 ions.append(m.groups()[:2])
148                 #append to ion list
149             else:
```

```
150                 rrngs.append(m.groups()[2:])
151                 #append to ranges list
152 
153     ions = pd.DataFrame(ions, columns=['number','name'])
154     #create pandas DataFrame of ions list
155     ions.set_index('number',inplace=True)
156     #sets the number index (wich is called with .ix or .iloc) as from the
        ↪  range file read
157     rrngs = pd.DataFrame(rrngs,
        ↪  columns=['number','lower','upper','vol','comp','colour'])
158     rrngs.set_index('number',inplace=True)
159 
160     rrngs[['lower','upper','vol']] =
        ↪  rrngs[['lower','upper','vol']].astype(float)
161     #float cast for the strings
162     rrngs[['comp','colour']] = rrngs[['comp','colour']].astype(str)
163 
164     return ions,rrngs
165 def label_ions(pos,rrngs):
166     """labels ions in a .pos or .epos dataframe (anything with a 'Da'
        ↪  column)
167     with composition and colour, based on an imported .rrng file."""
168 
169     pos['comp'] = ''
170     #adding a empty string coloumn as composition coloumn
171     pos['colour'] = '#FFFFFF'
172     #adding a colour coloumn, default #FFFFFF hex colour
173 
174     for n,r in rrngs.iterrows():
175         pos.loc[(pos.Da >= r.lower) & (pos.Da <=
            ↪  r.upper),['comp','colour']] = [r['comp'],'#' + r['colour']]
176         # ranging the atoms, and adding colour information from the range
            ↪  file
177     return pos
178 def deconvolve(lpos):
179     """Takes a composition-labelled pos file, and deconvolves
180     the complex ions. Produces a dataframe of the same input format
181     with the extra columns:
182        'element': element name
183        'n': stoichiometry
184     For complex ions, the location of the different components is not
185     altered - i.e. xyz position will be the same for several elements."""
186     #usually not used function
187 
188     import re
189 
```

```
190     out = []
191     pattern = re.compile(r'([A-Za-z]+):([0-9]+)')
192 
193     for g,d in lpos.groupby('comp'):
194         if g is not '':
195             for i in range(len(g.split(' '))):
196                 tmp = d.copy()
197                 cn = pattern.search(g.split(' ')[i]).groups()
198                 tmp['element'] = cn[0]
199                 tmp['n'] = cn[1]
200                 out.append(tmp.copy())
201     return pd.concat(out)
202 def getLattice(orig0, vecA, vecB, vecC, posAtoms, nA, nB, nC):
203     """ create a artificial lattice based on three translational symmetry
    ↪  vectors and
204     the so defines positions of the atoms within these three vectors vecA,
205     vecB, vecC. oring0 ... origin of the lattice in cartesian space. nA,
    ↪  nB, nC,
206     ... number of unit cells along corresponding cell vector, whole
    ↪  number of
207     unitcells nA*nB*nC. posAtoms ... positions relative to vecA, vecB and
    ↪  vecC
208     in the unitcell.
209     """
210     i = 0
211     j = 0
212     k = 0
213     #counting integers
214     numAtoms = np.size(posAtoms,0)
215     #number of atoms in cell
216 
217 #    print(numAtoms,'numAtoms')
218     AtomsList = np.zeros((nA*nB*nC*numAtoms,3))
219     #initializing atom positions as numpy array as zeros
220 #    print(posAtoms, 'posAtoms')
221 
222     while i < nA:
223         j=0
224         while j < nB:
225             k=0
226             while k < nC:
227                 l=0
228                 while l < numAtoms:
229                     orig = orig0 + i*vecA+j*vecB+k*vecC
230                     #current origin of unitcell
```

```
231                     AtomsList[i*nC*nB*numAtoms+j*nC*numAtoms+k*numAtoms+l,:
    ↪ ] = orig +
    ↪ posAtoms[l,0]*vecA+posAtoms[l,1]*vecB+posAtoms[l,2]
    ↪ *vecC
232                     #calculating absolut position in cartesian coordinates
233                     l+=1
234
235                 k+=1
236
237             j+=1
238
239         i+=1
240     return AtomsList
241 #def getLatticeNumpy(orig0, vecA, vecB, vecC, posAtoms, nA, nB, nC):
242 #    numAtoms = np.size(posAtoms,0)
243 #    print(numAtoms,'numAtoms')
244 #    AtomsList = np.zeros((nA*nB*nC*numAtoms,3))
245 #    mal = np.arange(0,nA)
246 #    print(mal*vecA)
247 #    print(mal)
248 #    AtomsList[i*nC*nB*numAtoms+j*nC*numAtoms+k*numAtoms+l,:] = orig +
    ↪ posAtoms[l,0]*vecA+posAtoms[l,1]*vecB+posAtoms[l,2]*vecC
249 #
250 #    return AtomsList
251 def isinKugel(x,y,z,punkt,r):
252     """ boolean function if x,y,z is in sphere around punkt with radius r.
253     """
254     if (x-punkt[0])**2+(y-punkt[1])**2 + (z-punkt[2])**2 <= r**2:
255         return True
256     else:
257         return False
258 def calcRDF(flannObj, params, FCC, rRDF, nBinRDF, target, stepSize = 1000):
259     """
260     calculates radia distribution function (RDF)
261     flannObj ... FLANN Oject (KD-Tree)
262     params ... params of FLANN Object
263     FCC ... source point set of the search for neighbors
264     rRDF ... radius within RDF is calculated
265     nBinRDF ... number of bins for histogram
266     target ... target point set
267     stepSize ... number of queries points which are sent to the
    ↪ pre-compiled
268     FLANN library
269     """
270     rRDFSq = rRDF**2
271     #calculating the squared distance, FLANN uses squared distances
```

```
272     rHist = np.zeros(nBinRDF)
273     #initialize histogram array
274     rvalues = np.linspace(0,rRDF,nBinRDF+1)
275     #create linear increasing radius values
276     rvaluesSq = np.power(rvalues,2)
277     #calculation of squared values of these r values
278     eps = 10**-10
279     #error limit, used to find exlude self-counting
280 
281     if np.size(FCC,0)% stepSize ==0:
282         folge = np.linspace(stepSize,np.size(FCC,0),(np.size(FCC,0)-stepSiz
            ↪  e)//stepSize+1)
283     elif np.size(FCC,0)<=stepSize:
284         folge = np.array([np.size(FCC,0)])
285     else:
286         folge = np.zeros((np.size(FCC,0)-stepSize)//stepSize+2)
287         folge = np.linspace(stepSize,((np.size(FCC,0)-stepSize)//stepSize+1
            ↪  )*stepSize,(np.size(FCC,0)-stepSize)//stepSize+1)
288         folge[-1] = np.size(FCC,0)
289     #Initializing folge which subdivides the query points depending on
290     #np.size(FCC,0)% stepSize ... number of query point modulo stepSize
291 
292     #now3 = time.time()
293     #timing for testing
294     targetSize =np.size(target,0)
295     #create targetSize, number of points in target point set
296 
297 #    targetSize = 10000
298     last = 0
299     #Initialize iteration integer for folge
300 
301     numberNN = 100
302     #initial number of nearest neighbor for query in FLANN
303     if numberNN > targetSize:
304         numberNN = targetSize
305         #change numberNN if targetSize is smalle than numberNN
306     for i in folge:
307         block = FCC[last:i,:]
308         #actual block of points in folge for query
309         countNumber = 1
310         #count integer for following queries
311         while np.size(block)>0:
312             # as long as number of query point in block > 0
313             numberNN*=countNumber
314             #adjust number of NN for query
315             if numberNN > targetSize:
```

```
316                 numberNN = targetSize
317                 #change numberNN if targetSize is smaller than numberNN
318                 #if numberNN > targetsize and is queried the library
                    ↪  crashes
319                 results, dists = flannObj.nn_index(block, numberNN,
                    ↪  checks=params['checks'])
320                 #query the numberNN nearest neighbors for the points in
                    ↪  block
321                 #usually stepSize*numberNN data points are returned for
                    ↪  results and dists
322                 #results is the index number in target, dists is the
                    ↪  calculated squared distance
323
324                 counts, histIndex = np.histogram(dists[dists>eps],
                    ↪  bins=rvaluesSq)
325                 #create histogram count of distances
326
327                 rHist += counts
328                 #add counts to final histogram
329                 break
330                 #breaks current while loop
331
332             results, dists = flannObj.nn_index(block, numberNN,
                ↪  checks=params['checks'])
333             #query the numberNN nearest neighbors for the points in block
334
335             block = block[dists[:,-1]<rRDFSq,:]
336             #take points which queried nearest neighbor is smaller than
                ↪  rRDFSq
337             #more NN are needed
338             countable = dists[dists[:,-1]>=rRDFSq,:]
339             #take points which last queried nearest neighbor is larger or
                ↪  equal than rRDFSq
340
341             counts, histIndex = np.histogram(countable[countable>eps],
                ↪  bins=rvaluesSq)
342             #create histogram count for already countable points
343             countNumber +=1
344             #increase loop counter
345
346 #             assert numberNN < targetSize, 'max targetsize'
347             #DEBUGGING info
348
349             rHist += counts
350             #add counts to final histogram
351
```

```
352         last = i
353         #last is current index for folge
354     return (rvalues[:-1]+rvalues[1:])/2, rHist
355     #return histogram values, r and final histogram counts
356 def getSDM(deltaXMax, deltaYMax, deltaZMax, bereich, binSDM=80):
357     """
358     Old SDM calculation function
359     deltaXMax ... x interval value for -deltaXMax to deltaXMax
360     deltaYMax ... y interval value
361     deltaZMax ... z interval value
362     binSDM ... number of bins for interval
363     """
364     edges0 = np.linspace(-deltaXMax,deltaXMax,binSDM)
365     edges1 = np.linspace(-deltaYMax,deltaYMax,binSDM)
366     edges2 = np.linspace(-deltaZMax,deltaZMax,binSDM)
367 
368     H, edges = np.histogramdd(2*deltaYMax*np.ones((1,3)), bins=(edges0,
        ↪  edges1, edges2))
369     #create threedimensional histogram
370     #usually assumed deltaYMax == deltaZMax == deltaXMax
371 
372     nx = np.int((np.max(bereich.x)-np.min(bereich.x))/deltaXMax)+1
373     ny = np.int((np.max(bereich.y)-np.min(bereich.y))/deltaYMax)+1
374     nz = np.int((np.max(bereich.z)-np.min(bereich.z))/deltaZMax)+1
375     #space is subdivides in blocks
376     #number of blocks in x,y,z direction
377 
378     origin0 = np.array([deltaXMax/2., deltaYMax/2., deltaZMax/2.])
379     #center of thirst block
380 
381     for i in range(nx):
382         for j in range(ny):
383             for k in range(nz):
384                 origin = origin0 + k*deltaXMax + j*deltaYMax + i*deltaXMax
385                 #current center of block
386                 H+= getSDMCubePd(deltaXMax, deltaYMax, deltaZMax,
                    ↪  bereich[cubePd(bereich, origin, deltaXMax, deltaYMax,
                    ↪  deltaZMax)].iloc[:,1:4], binSDM)[1]
387                 #calls getSDMCubePd for each block, and counts the
                    ↪  certain histogram counts into H
388                 #based on .read_epos(), .iloc[:,1:4] are the x,y,z
                    ↪  coordinates
389                 #cubePd returns datapoints within current block
390 
```

```
391 #                 H+= getSDMCube(deltaXMax, deltaYMax, deltaZMax,
    ↪ bereich[cubePd(bereich, origin, deltaXMax, deltaYMax,
    ↪ deltaZMax)].iloc[:,1:4].get_values(), binSDM)[1]
392
393     return edges, H
394 def getSDMCube(deltaXMax, deltaYMax, deltaZMax, bereich, binSDM=80):
395     """
396     Old version of getSDMCubePd() ?.
397     """
398     #    binSDM = 80
399     # has to be even
400     #OLD
401
402     deltas = np.zeros((np.size(bereich,0)*(np.size(bereich,0)-1)/2,3))
403
404     n = np.size(bereich,0)
405
406     counter = 0
407     edges0 = np.linspace(-deltaXMax,deltaXMax,binSDM)
408     edges1 = np.linspace(-deltaYMax,deltaYMax,binSDM)
409     edges2 = np.linspace(-deltaZMax,deltaZMax,binSDM)
410
411     for i,wert in enumerate(bereich.iloc[:,1:4]):
412         deltas[counter:counter+n-i-1,:] = wert - bereich[i+1:,:]
413         counter += n-i-1
414     #-------------------------------------------------------------------
    ↪ ----------
415
416     H, edges = np.histogramdd(deltas, bins=(edges0, edges1, edges2))
417     H += np.histogramdd(-deltas, bins=(edges0, edges1, edges2))[0]
418
419     return edges, H
420 def getSDMCubePd(deltaXMax, deltaYMax, deltaZMax, bereich, binSDM=80,
    ↪ binSizeZ = 80):
421     """
422     calculates three dimensional histogram of inter-atomic distances for
    ↪ each
423     combination of a pointset bereich.
424     Is called by getSDM().
425     """
426
427     deltas = np.zeros((np.size(bereich,0)*(np.size(bereich,0)-1)/2,3))
428     #create empy distance array
429     n = np.size(bereich,0)
430     #number of points
431
```

```
432     counter = 0
433     for i in xrange(n):
434 #         print(bereich.iloc[i,1:4])
435 #         print(bereich.iloc[i+1:,1:4])
436         #DEBUGGING info
437 
438         deltas[counter:counter+n-i-1,:] =bereich.iloc[i,:] -
            ↪  bereich.iloc[i+1:,:]
439         #calculates inter-atomic vectors for current to all following in
            ↪  the list
440         #Broadcasting Rule!
441 
442         counter += n-i-1
443         #update delta interating integer
444     #-------------------------------------------------------------------
      ↪  ----------
445 
446     edges0 = np.linspace(-deltaXMax,deltaXMax,binSDM)
447     edges1 = np.linspace(-deltaYMax,deltaYMax,binSDM)
448     edges2 = np.linspace(-deltaZMax,deltaZMax,binSizeZ)
449     #edges for histogram
450 
451     H, edges = np.histogramdd(deltas, bins=(edges0, edges1, edges2))
452     #calculate three dimensional histogram for deltas
453     H += np.histogramdd(-deltas, bins=(edges0, edges1, edges2))[0]
454     #add counts for exchanged source and target (-deltas)
455 
456     return edges, H
457 def getRotateZSDM(deltaZMax, deltas, phi, theta, binSDM=80):
458 #    binSDM = 80
459 #needs to be even
460 #-------------------------------------------------------------------------
    ↪  ------
461 # phi rotate around x axis, theta around y axis
462 # z is into the detector plane
463 #-------------------------------------------------------------------------
    ↪  ------
464 #psi311 = np.arctan(-pol311.det_y.mean()/flightPath)
465 #theta311 = np.arctan(pol311.det_x.mean()/flightPath)
466 #flightPath = 40. [mm]
467 #-------------------------------------------------------------------------
    ↪  ------
468 #informations of tests
469     """
470     calulate z-SDM for the rotation angles phi, theta, and inter-atomic
```

```
471     vectors deltas within -deltaZMax, deltaZMax with binSDM number of
    ↪   bins.
472     """
473
474     deltaZ = np.zeros(np.size(deltas,0))
475     edges2 = np.linspace(-deltaZMax,deltaZMax,binSDM)
476     stutz = (edges2[1:]+edges2[:-1])/2.
477
478     deltaZ = -np.sin(theta)*deltas[:,0] + np.cos(theta)*np.sin(phi)*deltas[
        ↪   :,1]+np.cos(theta)*np.cos(phi)*deltas[:,2]
479     #array of transformed delta z coordinates
480
481     z = np.histogram(deltaZ, bins = (edges2))[0]
482     #create histogram of delta z coordinates
483
484     return stutz, z
485 def getRotateXYSDM(deltaXMax, deltaYMax, deltas, phi, theta, deltaZ=0.,
    ↪   deltaDeltaZ=0.01, binSDM=100):
486     #    binSDM = 80
487     #needs to be even
488     #-----------------------------------------------------------------------
        ↪   ------
489
490     """
491     calculate XY SDM for given phi, theta rotation deltaZ position in
    ↪   three
492     dimensional histogram space for delta z values and deltaDeltaZ width
    ↪   around
493     it.
494     """
495
496     deltaXY = np.zeros((np.size(deltas,0),2))
497     deltasZ = np.zeros(np.size(deltas,0))
498     #initializing empty difference vectors "delta" arrays
499
500     edges0 = np.linspace(-deltaXMax,deltaXMax,binSDM)
501     edges1 = np.linspace(-deltaYMax,deltaYMax,binSDM)
502 #    stutz = (edges2[1:]+edges2[:-1])/2.
503
504     deltaXY[:,0] = np.cos(theta)*deltas[:,0] + np.sin(theta)*np.sin(phi)*de
        ↪   ltas[:,1]+np.sin(theta)*np.cos(phi)*deltas[:,2]
505     #tranformed delta x coordinates
506
507     deltaXY[:,1] = np.cos(phi)*deltas[:,1] -np.sin(phi)*deltas[:,2]
508     #tranformed delta y coordinates
```

```
509     deltasZ = -np.sin(theta)*deltas[:,0] + np.cos(theta)*np.sin(phi)*deltas
        ↪  [:,1]+np.cos(theta)*np.cos(phi)*deltas[:,2]
510     #tranformed delta z coordinates
511
512     xySDM = np.logical_and(deltasZ <= deltaZ+deltaDeltaZ, deltasZ >=
        ↪  deltaZ-deltaDeltaZ)
513     #selecting subset via logical array
514
515     H, edges = np.histogramdd(deltaXY[xySDM,:], bins=(edges0, edges1))
516     #creating two-dimensional histogram for chosen inter-atomic vectors
517
518     stutz =
        ↪  [(edges[0][:-1]+edges[0][1:])/2.,(edges[1][:-1]+edges[1][1:])/2.]
519     #edges in x and y direction
520
521     return stutz, H
522 def getRotateVec(deltas, phi, theta, gamma):
523     # phi rotating around x axis, theta around y axis
524     # z into the detector plane
525     """
526     calculates rotated vectors from deltas vectors, gamma not used anymore
527     phi, theta as rotation angles
528     """
529
530     deltaXYZ = np.zeros((np.size(deltas,0),3))
531     #Initialize array to return
532
533
534     deltaXYZ[:,0] = np.cos(theta)*deltas[:,0] + np.sin(theta)*np.sin(phi)*d
        ↪  eltas[:,1]+np.sin(theta)*np.cos(phi)*deltas[:,2]
535     #calculate x coordinates
536     deltaXYZ[:,1] = np.cos(phi)*deltas[:,1] -np.sin(phi)*deltas[:,2]
537     #calculate y coordinates
538     deltaXYZ[:,2] = -np.sin(theta)*deltas[:,0] + np.cos(theta)*np.sin(phi)*
        ↪  deltas[:,1]+np.cos(theta)*np.cos(phi)*deltas[:,2]
539     #calculate z coordinates
540
541     return deltaXYZ
542 def getRotateXYSDMAll(deltaXMax, deltaYMax, deltas, phi, theta, binSDM=100):
543     #    binSDM = 80
544     #needs to be even
545     #-------------------------------------------------------------------
        ↪  ------
546     """
547     calculates the XY SDM for all difference vectors deltas, without
    ↪  choosing
```

```
548     a subset (usually a peak in the z SDM) as in getRotateXYSDM().
549     """
550 
551     deltaXY = np.zeros((np.size(deltas,0),2))
552     #Initialize vector for difference vectors in XY
553 
554     edges0 = np.linspace(-deltaXMax,deltaXMax,binSDM)
555     edges1 = np.linspace(-deltaYMax,deltaYMax,binSDM)
556     #edges for histogram
557 
558     deltaXY[:,0] = np.cos(theta)*deltas[:,0] + np.sin(theta)*np.sin(phi)*de
        ↪ ltas[:,1]+np.sin(theta)*np.cos(phi)*deltas[:,2]
559     deltaXY[:,1] = np.cos(phi)*deltas[:,1] -np.sin(phi)*deltas[:,2]
560     #calculate rotated x and y coordinates
561 
562     H, edges = np.histogramdd(deltaXY, bins=(edges0, edges1))
563     #create 2D histogram
564 
565     stutz =
        ↪ [(edges[0][:-1]+edges[0][1:])/2.,(edges[1][:-1]+edges[1][1:])/2.]
566 
567 
568     return stutz, H
569 def getRotateXYSDMhex(deltaXMax, deltaYMax, deltas, phi, theta, deltaZ=0.,
    ↪ deltaDeltaZ=0.01):
570     #    binSDM = 80
571     #needs to be even
572     #------------------------------------------------------------------
        ↪ ------
573     """
574     calulates XY SDM for chosen subset as in getRotateXYSDM(), returns
    ↪ values
575     as needed or the hex plot in matploblib.pyplot.
576     """
577 
578     deltaXY = np.zeros((np.size(deltas,0),2))
579     deltasZ = np.zeros(np.size(deltas,0))
580 
581     deltaXY[:,0] = np.cos(theta)*deltas[:,0] + np.sin(theta)*np.sin(phi)*de
        ↪ ltas[:,1]+np.sin(theta)*np.cos(phi)*deltas[:,2]
582     deltaXY[:,1] = np.cos(phi)*deltas[:,1] -np.sin(phi)*deltas[:,2]
583     deltasZ = -np.sin(theta)*deltas[:,0] + np.cos(theta)*np.sin(phi)*deltas
        ↪ [:,1]+np.cos(theta)*np.cos(phi)*deltas[:,2]
584 
585     xySDM = np.logical_and(deltasZ <= deltaZ+deltaDeltaZ, deltasZ >=
        ↪ deltaZ-deltaDeltaZ)
```

```
586
587     return deltaXY[xySDM,:]
588 def getDeltasSDM(bereich):
589     """
590     calculates difference vectors
591     including the so generated inverted difference vectors will give all
592     inter-atomic difference vectors
593     """
594     deltas = np.zeros((np.size(bereich,0)*(np.size(bereich,0)-1)/2,3))
595     #Initializing array for difference vectors
596     n = np.size(bereich,0)
597     #number of points
598
599     counter = 0
600     for i in xrange(n):
601 #         print(bereich.iloc[i,1:4])
602 #         print(bereich.iloc[i+1:,1:4])
603     #DEBUGGING Info
604         deltas[counter:counter+n-i-1,:] = bereich.iloc[i,:] -
        ↪  bereich.iloc[i+1:,:]
605         #calculate difference vectors from ith point to all following in
        ↪  the list
606         #broadcasting
607         counter += n-i-1
608         #update counter
609
610     return deltas
611 def getDeltasSDMLarge(bereich, NN):
612     """
613     currently used
614     calculates the difference vectors for a given number of nearest
    ↪  neighbors
615     NN, for given point set bereich.
616     """
617     last = 0
618     #initialize a counting integer
619     stepSize = 10000
620     #number of points to query into FLANN
621     deltas = np.zeros((np.size(bereich,0)*NN,3))
622     #Initialization of difference vector array
623
624     flannObj = FLANN()
625     #create a FLANN object
626     params = flannObj.build_index(bereich, algorithm=4,
        ↪  target_precision=1., log_level = "info") #algorithm=4 selects
        ↪  kdtree single
```

```
627     #set the parameters for FLANN object kdtree single means an exact
        ↪  kdtree
628 
629     if np.size(bereich,0)% stepSize ==0:
630         folge = np.linspace(stepSize,np.size(bereich,0),(np.size(bereich,0)
            ↪  -stepSize)//stepSize+1)
631     elif np.size(bereich,0)<=stepSize:
632         folge = np.array([np.size(bereich,0)])
633     else:
634         folge = np.zeros((np.size(bereich,0)-stepSize)//stepSize+2)
635         folge = np.linspace(stepSize,((np.size(bereich,0)-stepSize)//stepSi
            ↪  ze+1)*stepSize,(np.size(bereich,0)-stepSize)//stepSize+1)
636         folge[-1] = np.size(bereich,0)
637     #subdividing the points into block of points (stepSize) to query FLANN
638 
639     for i in folge:
640         block = bereich[last:i,:]
641         #take a block of points
642         results, dists = flannObj.nn_index(block, NN+1,
            ↪  checks=params['checks'])
643         #query distances for NN neares neighbors (NN+1) due to the fact
            ↪  that
644         #points "see" themselves
645 
646 #         print(i,last,NN)
647 #         print(np.size(bereich[results[1:,:]],0))
648         #DEBUGGING Info
649 
650         arr = bereich[results[:,0]]
651         #take points from bereich which are found results[:,0]
652         arr = arr[:,np.newaxis]
653         #needed for right size of numpy array
654         deltas[last*NN:i*NN,:] =
            ↪  np.reshape(bereich[results[:,1:]]-arr,((i-last)*(NN),3))
655         #calculating the difference vectors and putting it into the
            ↪  difference
656         #vector array, broadcasting!
657         last = i
658         #update counter
659     flannObj.delete_index()
660     #delete flann object to free memory
661     return deltas
662 def getZSDM(edges, H):
663     """
664     should calculate the z SDM based on a 3 dimensional histogram
665     """
```

```
666     stutz = (edges[2][1:]+edges[2][:-1])/2.
667     z = np.sum(np.sum(H,axis=0),axis=0)
668     #sums over x and then y axes
669     return stutz, z
670 def plotSDM(edges, H, deltaZ = 0., deltaDeltaZ = 0.05, text =''):
671     """
672     returns a figure with z SDM and XY SDM plotted (for chosen subset of
    ↪ points),
673     based on three dimensional histogram of difference vectors.
674     """
675     stutz = (edges[2][1:]+edges[2][:-1])/2.
676     xySDM = np.logical_and(edges[2][:] <= deltaZ+deltaDeltaZ,edges[2][:] >=
    ↪  deltaZ-deltaDeltaZ)
677     #choosing the subset of points
678 #    print(xySDM)
679     #DEBUGGING Info
680
681     X,Y = np.meshgrid((edges[0][:-1]+edges[0][1:])/2.,(edges[1][:-1]+edges[
    ↪  1][1:])/2.)
682     #create a meshgrid based on x,y edges of the histogram
683
684     fig = plt.figure()
685     #create figure object
686     fig.set_size_inches(10,10.)
687
688     ax = fig.add_subplot(221)
689     #adding a subplot to figure object
690
691     ax.axvline(deltaZ-deltaDeltaZ,color='r',linewidth=2.)
692     #vertical line plotting in red for the chosen subset in the z SDM
693     ax.axvline(deltaZ+deltaDeltaZ,color='r',linewidth=2.)
694     #vertical line plotting in red for the chosen subset in the z SDM
695     ax.plot(stutz, np.sum(np.sum(H,axis=0),axis=0))
696     #plot z SDM
697     ax.grid(True)
698     ax.set_title(text)
699
700     ax2= fig.add_subplot(222)
701     #adding another subplot to figure object
702     ax2.contourf(X,Y,np.sum(H[:,:,xySDM],axis=2),cmap=cm.gray)
703     #contourplot of XY SDM, with chosen colormap
704     ax2.set_title('XY-SDM')
705     return fig
706 def createNNHist(numberNN, data, flann0, paramFl, ranges, binsize):
707     """
708     return neares neighbor histogram
```

```
709     numberNN ... number of nearest neighbor
710     data ... point set
711     flann0 ... FLANN object
712     paramFl --- corresponding parameters for FLANN object
713     """
714     eps = 10**-5
715     #epsilon value which is used to avoid selfcounting
716 
717     resultN, distsN = flann0.nn_index(data, numberNN+1,
        ↪  checks=paramFl['checks'])
718     #FLANN query for number of nearest neigbors (numberNN), numberNN+1
        ↪  used
719     #to later avoid selfcounting
720     distsN = np.sqrt(distsN)
721     #calculate sqare root of distances, FLANN uses squared distances
722 
723 #    d = distsN[:,-1]
724     #DEBUGGING Info
725 
726     dists = np.zeros(np.size(distsN,0))
727 
728 #    print(distsN.shape)
729     #DEBUGGING Info
730 
731     groesser = distsN[:,0] > eps
732     kleiner = distsN[:,0] <= eps
733     #splitset into within epsilon value kleiner, and outside groesser
734     #split is needed if flann Object is not built on the point set from
        ↪  data
735 
736     dists[kleiner] = distsN[kleiner,-1]
737     #takes last coloumn if smallest distance is in epsilon value for
        ↪  example if Si-Si
738     dists[groesser] = distsN[groesser,-2]
739     #takes second last coloumn if smallest distance outside the epsilon
        ↪  value,
740     #last value would be numberNN+1'th nearest neighbor, for example if
        ↪  Si-Mg
741 
742 #    assert np.size(dists[dists<eps]) == 0, str(dists[dists<eps]) + " " +
    ↪  str(distsN[dists==0.])
743     #DEBUGGING Info
744     assert np.size(dists[dists<eps]) == 0, str(dists[dists<eps]) + " " +
        ↪  str(distsN[dists<eps])
745     #raises error if assertion is not fullfilled
746     return np.histogram(dists, bins=binsize, range=ranges)
```

```
747 def createNNHistAllTill(numberNN, data, flann0, paramFl):
748     """
749     return neares neighbor histogram
750     numberNN ... number of nearest neighbor
751     data ... point set
752     flann0 ... FLANN object
753     paramFl --- corresponding parameters for FLANN object
754     """
755     eps = 10**-5
756     #epsilon value which is used to avoid selfcounting
757 
758     resultN, distsN = flann0.nn_index(data, numberNN+1,
        ↪  checks=paramFl['checks'])
759     #FLANN query for number of nearest neigbors (numberNN), numberNN+1
        ↪  used
760     #to later avoid selfcounting
761     distsN = np.sqrt(distsN)
762     #calculate sqare root of distances, FLANN uses squared distances
763 
764 #     d = distsN[:,-1]
765     #DEBUGGING Info
766 
767     dists = np.zeros((np.shape(distsN)[0],np.shape(distsN)[1]-1))
768 #     print(np.shape(distsN))
769 #     print(np.shape(distsN))
770 
771     groesser = distsN[:,0] > eps
772     kleiner = distsN[:,0] <= eps
773     #splitset into within epsilon value kleiner, and outside groesser
774     #split is needed if flann Object is not built on the point set from
        ↪  data
775 
776     dists[kleiner] = distsN[kleiner,1:]
777     #takes last coloumn if smallest distance is in epsilon value for
        ↪  example if Si-Si
778     dists[groesser] = distsN[groesser,:-1]
779     #takes second last coloumn if smallest distance outside the epsilon
        ↪  value,
780     #last value would be numberNN+1'th nearest neighbor, for example if
        ↪  Si-Mg
781 #     assert np.size(dists[dists<eps]) == 0, str(dists[dists<eps]) + " " +
    ↪  str(distsN[dists<eps])
782     #raises error if assertion is not fullfilled
783     return dists
784 def getMean(a):
785     """
```

```
786     searches the index of half greater equal half the sum of the array
787     """
788     summe = np.sum(a)
789     #returns sum of all elements of a
790     index = 0
791     erg = 0
792     counter =0
793     for i in a:
794         if erg>=summe/2.:
795             index = counter
796             break
797         erg +=i
798         counter+=1
799     return index
800 def getComp(lpos):
801     """
802     prints composition
803     """
804     comp = []
805     name = []
806     for g,d in lpos.groupby('comp'):
807         #.groupby('comp') returns iterateable object, grouped by equal
            ↪  'comp'
808         #labeling
809         comp.append(len(d))
810         #number of labeled composition in lpos DataFrame appended to a
            ↪  list
811         name.append(g)
812         #name of species appended to name list
813     comp = np.array(comp)
814     #overwrites list with numpy array created from comp list
815     comp = comp/np.float(np.sum(comp))
816     #norming, float cast needed otherwise integer division executed
817     print('ion %')
818     for i, wert in enumerate(name):
819         print(wert+" " + str(100.*comp[i]))
820         #print composition in percentage
821     return name, comp
822 def saveComp(lpos, path):
823     """
824     prints composition and saves .txt file to path, see getComp()
825     """
826     comp = []
827     name = []
828     output = ""
829     for g,d in lpos.groupby('comp'):
```

```
830             comp.append(len(d))
831             name.append(g)
832         comp = np.array(comp)
833         comp = comp/np.float(np.sum(comp))
834         output += 'ion %\n'
835         print('ion %')
836         for i, wert in enumerate(name):
837             print(wert+" " + str(100.*comp[i]))
838             output += wert + " " + str(100.*comp[i]) + "\n"
839         f = open(path,'w')
840         f.write(output)
841         f.close()
842         return
843     def getCompEl(name, comp, such):
844         """
845         function to sum over composition values where the name is a subset of
    ↪    such,
846         for example: should sum compositions of 'Al:1' and 'Al:1 H:1' if 'Al'
    ↪    is such
847         """
848         sum0=0
849         for i, wert in enumerate(name):
850             if wert.find(such) > -1:
851                 sum0 += comp[i]
852         print(such, sum0)
853         return sum0
854     def cube(FCC, origin, deltaXMax, deltaYMax, deltaZMax):
855         """
856         function to select datapoint within a block defined by
857         plusminus deltaXMax, deltaYMax and deltaZMax
858         """
859         bereich = FCC[FCC[:,0]<=origin[0] + deltaXMax,:]
860         bereich = bereich[bereich[:,0]>=origin[0] - deltaXMax,:]
861 
862         bereich = bereich[bereich[:,1]<=origin[1] + deltaYMax,:]
863         bereich = bereich[bereich[:,1]>=origin[1] - deltaYMax,:]
864 
865         bereich = bereich[bereich[:,2]<=origin[2] + deltaZMax,:]
866         bereich = bereich[bereich[:,2]>=origin[2] - deltaZMax,:]
867 
868         return bereich
869     def cubePd(FCC, origin, deltaXMax, deltaYMax, deltaZMax):
870         """
871         function to select datapoint within a block defined by
872         plusminus deltaXMax, deltaYMax and deltaZMax for the pandas DataFrame
    ↪    pos
```

```
873         format
874         """
875         bereich = FCC.x<=origin[0] + deltaXMax
876         bereich = np.logical_and(FCC.x>=origin[0] - deltaXMax,bereich)
877 
878         bereich = np.logical_and(FCC.y<=origin[1] + deltaYMax, bereich)
879         bereich = np.logical_and(FCC.y>=origin[1] - deltaYMax, bereich)
880 
881         bereich = np.logical_and(FCC.z<=origin[2] + deltaZMax, bereich)
882         bereich = np.logical_and(FCC.z>=origin[2] - deltaZMax, bereich)
883 
884         return bereich
885     def coordToView(bereich):
886         """
887         creates DataFrame object which can be viewed with volvis(), if only
    ↪    x,y,z
888         data is available (bereich), for example for an artificial lattice.
889         """
890         pos = pd.DataFrame({'x': bereich[:,0],
891                             'y': bereich[:,1],
892                             'z': bereich[:,2],
893                             'Da': np.ones(np.size(bereich[:,0]))})
894         ions, rrngs = read_rrng('example-data/rangefile.rrng')
895         lpos = label_ions(pos,rrngs)
896 
897         return lpos
898     def getFCC(origin = np.array([0., 0., 0.]), n=50, aAl = 0.404):
899         """
900         return a artificial fcc lattice with primitive lattice constant aAl
    ↪    and
901         n unit cells
902         """
903         vectorA = aAl * np.array([1, 0., 0.])
904         vectorB = aAl * np.array([0., 1., 0.])
905         vectorC = aAl * np.array([0., 0., 1.])
906         #translational vectors of unitcell
907 
908         point = np.zeros((4,3))
909         #initial point array
910 
911         point[0,:]=[0., 0., 0.]
912         point[1,:]=[0.5, 0.5, 0.]
913         point[2,:]=[0., 0.5, 0.5]
914         point[3,:]=[0.5, 0., 0.5]
915         #atom positions within chosen vector set
916 
```

```
917     FCC = getLattice(origin, vectorA, vectorB, vectorC, point, n, n, n)
918     #call getLattice with chosen parameters
919     return FCC
920 #def getMean(a):
921 #     summe = np.sum(a)
922 #     index = 0
923 #     erg = 0
924 #     counter =0
925 #     for i in a:
926 #         if erg>=summe/2.:
927 #             index = counter
928 #             break
929 #         erg +=i
930 #         counter+=1
931 #     return index
932     #Outcommented overrode former getMean() ?
933 def getDrehMatrix(psi, theta, phi):
934     """
935     should create a rotation matrix
936     seems unfinished, see dreheCordPd(),dreheCordPd2()
937     """
938     rueck = np.matrix([[np.cos(psi)*np.cos(phi)-np.sin(psi)*np.cos(theta)*n
      ↪  p.sin(phi),
      ↪  -np.cos(psi)*np.sin(phi)-np.sin(psi)*np.cos(theta)*np.cos(phi),
      ↪  np.sin(psi)*np.sin(theta)],
939                        [np.sin(psi)*np.cos(phi)+
                           ↪  np.cos(psi)*np.cos(theta)*np.sin(phi),
                           ↪  -np.sin(psi)*np.sin(phi)+np.cos(psi)*np.cos(thet
                           ↪  a)*np.cos(phi),
                           ↪  -np.cos(psi)*np.sin(theta)],
940                         [np.sin(theta)*np.sin(phi),
                            ↪  np.sin(theta)*np.cos(phi), np.cos(theta)]])
941     #rotation matrix
942     rueck = rueck.transpose()
943     #transposes it and overwrites former
944     return(rueck)
945 def getDrehMatrixCardan(alpha, beta, gamma):
946     """
947     should create a Cardan rotation matrix
948     seems unfinished, see dreheCordPd(),dreheCordPd2()
949     """
950     D1 = np.matrix([[1., 0., 0.,],
951                     [0., np.cos(alpha), -np.sin(alpha)],
952                      [0., np.sin(alpha), np.cos(alpha)]])
953     D2 = np.matrix([[np.cos(beta), 0., np.sin(beta)],
954                      [0., 1., 0.],
```

```
955                     [-np.sin(beta), 0., np.cos(beta)]])
956     D3 = np.matrix([[np.cos(gamma), -np.sin(gamma), 0.],
957                     [np.sin(gamma), np.cos(gamma), 0.],
958                      [0.,0.,1.]])
959     D = D1*D2*D3
960     #should be the matrix product of the three matrices
961     D = D.transpose()
962     #transposes and overwrites it
963     return(D)
964 def dreheCordPd(lpos, psi, theta, phi):
965     #something is wrong here
966     rueck = lpos.copy()
967     rueck.iloc[:,1:4]=
        ↪  np.array(lpos.iloc[:,1:4].get_values()*getDrehMatrix(psi,theta,phi))
968     return rueck
969 def dreheCordPd2(lpos, psi, theta, phi):
970     #something is wrong here
971     rueck = lpos.copy()
972     rueck.iloc[:,1:4]= np.array(lpos.iloc[:,1:4].get_values()*getDrehMatrix
        ↪  Cardan(psi,theta,phi))
973     #should transform the coordinates,
974     return rueck
975 def getRasterwinkel(deltas, startPsi, startTheta, binSize, deltaBereich=1.,
    ↪  maxAnglePsi = np.pi/4., maxAngleTheta = np.pi/4., teilungPsi = 200.,
    ↪  teilungTheta = 200.):
976     """
977     scanning the maximum height of the z SDMs for various psi and theta
    ↪  angles
978     around center angled startPsi, startTheta
979     maxAnglePsi ... maximum scanned plusminus psi angle range
980     maxAngleTheta ... maximum scanned plusminus theta angle range
981     teilungPsi ... number of subdivisions in psi
982     teilungTheta ... number of subdivisions in theta
983     startPsi ... center psi angle
984     startTheta ... center theta angle
985     binSize ... bin size for z SDM
986     deltas ... array of difference vectors
987     """
988     deltaPsi = np.arange(-maxAnglePsi+startPsi, maxAnglePsi+startPsi,
        ↪  2*maxAnglePsi/teilungPsi)
989     deltaTheta = np.arange(-maxAngleTheta+startTheta,
        ↪  maxAngleTheta+startTheta, 2*maxAngleTheta/teilungTheta)
990     #creates points in psi and theta angle space
991
992     fig2 = plt.figure()
993     conv = 2.54
```

```
994     #1 inch = 2.54cm
995     breite=15/conv
996     hoehe=24/conv
997     fig2.set_size_inches(breite,hoehe)
998     #creates and sets size of figure
999
1000    ax3 = fig2.add_subplot(211)
1001    ax2 = fig2.add_subplot(212)
1002    #adds subplots to figure
1003
1004    psi = 0.
1005    theta = 0.
1006    #initialize variables for later holding the coordinates of the
        ↪ maximum found
1007    maxZSDM = 0
1008    #height of the maximum z SDM
1009    rasterWinkel = np.zeros((np.size(deltaPsi), np.size(deltaTheta)))
1010    #initialize 2d array for holding found max z SDM
1011    #-------------------------------------------------------------
        ↪ ----------------------
1012    for i,wert in enumerate(deltaPsi):
1013        for j, wert2 in enumerate(deltaTheta):
1014            stutz, z = getRotateZSDM(deltaBereich, deltas, wert, wert2,
                ↪ binSDM = binSize)
1015            #calculates z SDM
1016            rasterWinkel[i,j] = np.max(z)
1017            #saves maximum into 2d array
1018            if np.max(z)>maxZSDM:
1019                maxZSDM = np.max(z)
1020                psi = wert
1021                theta = wert2
1022            #updates coordinates if greater than previous maximum in
                ↪ scanned z SDMs
1023    X,Y = np.meshgrid(deltaPsi,deltaTheta)
1024    #creates meshgrid for contour plotting
1025
1026    CS = ax3.contourf(X, Y, rasterWinkel.transpose())
1027    #plots contour plot
1028    plt.colorbar(CS)
1029    #adds a colorbar
1030    #-------------------------------------------------------------
        ↪ ----------------------
1031    ax2.plot(*getRotateZSDM(2*deltaBereich, deltas, psi, theta, binSDM =
        ↪ binSize))
1032    #plots the z SDM of the maximum z SDM found in the psi theta range
1033    return psi, theta, deltaPsi, X,Y, rasterWinkel, fig2
```

```
1034 def smoothingZSDM(z, maxIter):
1035     """
1036     smooths a function intended to be used for z SDM smoothing, see paper
     ↪  from Moody
1037     """
1038     z = z.copy()
1039     #needed,du to that i-1 old indices are needed
1040     for j in xrange(maxIter):
1041         z0 = z.copy()
1042         for i in xrange(1, np.size(z)-1):
1043     #         print(i)
1044     #DEBUGGING Info
1045             z[i] = min(z0[i], (z0[i+1]+z0[i-1])/2.)
1046     return z
1047 def smoothingXYSDM(z, maxIter):
1048     """
1049     smooths a XY SDM, see paper from Moody
1050     unfinished?
1051     """
1052     z = z.copy()
1053     for j in xrange(maxIter):
1054         z0 = z.copy()
1055         for i in xrange(1,np.size(z,0)-1):
1056             for k in xrange(1,np.size(z,1)-1):
1057                 z[i,k] = min(z[i,k],(z[i-1,k] + z[i+1,k] + z[i,k-1]
                     ↪  +z[i,k+1])/4.)
1058                 #should it be z0?
1059     return z
1060 def getPeaks(xs, zInter, minDistance = 0.02):
1061     """
1062     should search for local maxima and lists them into peaks list if it
     ↪  is
1063     larger away than minDistance from the minimum of xs entry.
1064     xs ... delta z values of the z SDM (abscissa in plot)
1065     zInter ... counts of z SDM (ordinate in plot)
1066     """
1067     peaks = []
1068     peaksDist0 = np.min(xs)
1069 
1070     for i in range(1, np.size(zInter)-1):
1071         if zInter[i] > zInter[i-1] and zInter[i]>zInter[i+1] and
          ↪  (xs[i]-peaksDist0)> minDistance:
1072             peaks.append(i)
1073             peaksDist0 = xs[i]
1074     return peaks
1075 def getLayerDistance(xs, zInter, peaks, maxNullAbw = -0.1):
```

```
1076        posPeaks =xs[peaks][xs[peaks]>maxNullAbw]
1077        #get positive delta z values of identified peaks, see getPeaks()
1078
1079        summe = 0
1080        for i in range(np.size(posPeaks)):
1081            if i>0:
1082                summe += (posPeaks[i]-posPeaks[0])/i
1083        #         print((posPeaks[i]-posPeaks[0])/i)
1084                #DEBUGGING Info
1085        #calculates the difference in delta z values for neighboring peaks,
1086        #which should be the inter planar distance
1087
1088        averageLayerDistToOrigin = summe /(np.size(posPeaks)-1)
1089        #means the distances
1090
1091        summe = 0
1092        count = 0
1093        for i in xrange(1,np.size(posPeaks)):
1094            summe += (posPeaks[i]-posPeaks[i-1])
1095            count+=1
1096
1097        averageLayerDistDiff = summe /(np.size(posPeaks)-1)
1098        print(averageLayerDistDiff)
1099        averageLayerDistDiff = summe /(count)
1100        print(averageLayerDistDiff)
1101        #DEBUGGING
1102
1103        return averageLayerDistToOrigin, averageLayerDistDiff
1104    def getLatticeConstant(dhkl, hkl):
1105        """
1106        should calculate the lattice constant of a cubic lattice by
        ↪ parameters interplanar
1107        distance dhkl and miller index of the plane
1108        """
1109        a = dhkl*np.sqrt(np.dot(hkl,hkl))
1110        return a
1111    def getCoresBool(lpos, elementCore):
1112        """
1113        returns atoms of defined list of species (elementCore),
1114        lpos ... pandas DataFrame of atom probe data
1115        there is a more elegant way to do this alread for the DataFrame
        ↪ object...
1116        compare for example lpos.loc[:,[('comp' == 'species1:1')|('comp' ==
        ↪ 'species2:1')]]
1117        """
1118        rueck = np.zeros(np.size(lpos,0), dtype = bool)
```

```
1119        for i in elementCore:
1120            rueck = np.logical_or(rueck, lpos.comp == i)
1121
1122        return rueck
1123    def getClusterAtomsL(dmax, dlink, derode, K, lpos, elementCore):
1124        """
1125        should reproduce core-linkage or maximum separation cluster
        ↪ identification method (if both have derode = 0)
1126        dmax, dlink, derode, K ... parameters for cluster identification,
        ↪ squared distances (for dmax, dlink, derode) must
1127        be given for example if dmax should be 0.74, 0.74**2 must be put as
        ↪ parameter when calling
1128        getClusterAtomsL().
1129        """
1130        #-------------------------------------------------------------------
        ↪ ----------
1131        print([dmax,dlink,derode,K],['dmax','dlink','derode','K'])
1132        #DEBUGGING Info
1133
1134        coreBool = getCoresBool(lpos, elementCore)
1135        #get specified logical array
1136        dataCluster = lpos.loc[coreBool,['x','y','z']].get_values() # .ix
        ↪ replaced
1137        #take subset of DataFrame Object, x,y,z coordinates of chosen core
        ↪ atoms
1138
1139        print(dataCluster.shape)
1140        #DEBUGGING Info
1141
1142        set_distance_type('euclidean')
1143        flannCluster = FLANN()
1144        #creates FLANN object
1145
1146        NN = 100
1147        #starting nearest neighbor number
1148
1149        paramsCluster = flannCluster.build_index(dataCluster,algorithm=4,
        ↪ target_precision=1., log_level = "info"); #algorithm=4 selects
        ↪ kdtree single
1150        #builds kd tree
1151        #------------------------------------ core step
1152        results, dists = flannCluster.nn_index(dataCluster, K+1,
        ↪ checks=paramsCluster['checks'])
1153        #queries K+1 nearest neighbor distances, K+1 , due to that atoms see
        ↪ theirself
1154        resultsCore = results[:,0][dists[:,-1]<=dmax]
```

```
1155        #identify core atoms, take the atoms which Kth nearest neighbor is
            ↪  smaller or equal dmax
1156
1157        print('resultsCore ',resultsCore.shape)
1158        #DEBUGGING Info
1159        assert np.size(resultsCore) > 1, 'no core atoms identified'
1160        #raises error if no core atoms are found
1161        resultsCore = np.unique(resultsCore.flatten())
1162        #returns a uniqe set
1163
1164        flannCluster.delete_index()
1165        #deletes built index
1166        print('cores ',resultsCore.shape)
1167        #DEBUGGING Info
1168        #-------------------------------------------------------------------
            ↪  ---------- linkage
            ↪  distance
1169        positionen = lpos.loc[:,['x','y','z']].get_values()
1170        #take positions of all atoms
1171        stepSize = 10000
1172        results0 = np.array([], dtype = np.int32)
1173
1174        paramsCluster = flannCluster.build_index(positionen, algorithm=4,
            ↪  target_precision=1., log_level = "info"); #algorithm=4 selects
            ↪  kdtree single
1175        #builts kdtree
1176
1177        FCC = np.array(lpos.loc[coreBool,['x','y','z']].get_values()[resultsCor
            ↪  e,:], order='C') # ix
            ↪  replaced
1178        #creates array to use as query points
1179        print('FCC ',FCC.shape)
1180        #DEBUGGING Info
1181
1182        if np.size(FCC,0)% stepSize ==0:
1183            folge = np.linspace(stepSize,np.size(FCC,0),(np.size(FCC,0)-stepSiz
                ↪  e)//stepSize+1)
1184            print('if')
1185            #DEBGUGGING Info
1186        elif np.size(FCC,0)<=stepSize:
1187            folge = np.array([np.size(FCC,0)])
1188            print('elif')
1189            #DEBUGGING Info
1190        else:
1191            folge = np.zeros((np.size(FCC,0)-stepSize)//stepSize+2)
```

```
1192            folge = np.linspace(stepSize,((np.size(FCC,0)-stepSize)//stepSize+1
                ↪  )*stepSize,(np.size(FCC,0)-stepSize)//stepSize+1)
1193            folge[-1] = np.size(FCC,0)
1194            print('else')
1195            #DEBUGGING Info
1196        #subdividing point set for query
1197
1198        print(folge)
1199        #DEBUGGING Info
1200        targetSize =np.size(positionen,0)
1201        last = 0
1202        if NN > targetSize:
1203            NN = targetSize
1204        for i in folge:
1205            block = FCC[last:i,:]
1206            #take a block of the point set to query
1207            print('block',block.shape)
1208            #DEBUGGING Info
1209            dists = np.zeros((2,2))
1210            while np.min(dists[:,-1]) <= dlink:
1211                results, dists = flannCluster.nn_index(block, NN,
                    ↪  checks=paramsCluster['checks'])
1212                #query points around block while their maximum distance is
                    ↪  smaller than dlink
1213                if np.min(dists[:,-1]) > dlink:
1214                    break
1215                NN+=10
1216                if NN > targetSize:
1217                    NN = targetSize
1218            results0=np.union1d(results0, results[dists<=dlink].flatten())
1219            #create unique set, avoids doubled indices
1220            last = i
1221        print('linking', results0.shape)
1222        #DEBUGGING Info
1223        #-------------------------------------------------------------------
            ↪  ----------
1224        #for the core linkage method here would it be necessary to identify
            ↪  the cluster numbers
1225        #for reasons of speed this is here done after the eroding step
1226        #only for derode = 0 therefore the same results are expected
1227        resultsErode = results0
1228
1229        del results0
1230        del dists
1231        del results
1232        #free memory
```

```
1233
1234        mask = np.zeros(np.size(lpos,0), dtype = bool)
1235        mask[resultsErode] = True
1236        #creates a logical array, sets indices True for identified core and
            ↪  linked atoms
1237
1238        flannCluster.delete_index()
1239        #delete kdtree
1240
1241        stepSize = 10000
1242        results0 = np.array([], dtype =np.int32)
1243
1244        paramsCluster = flannCluster.build_index(positionen[mask,:],
            ↪  algorithm=4, target_precision=1., log_level = "info");
            ↪  #algorithm=4 selects kdtree single
1245        #create kd tree on those atoms
1246        FCC2 = np.array(positionen[np.logical_not(mask),:], order='C')
1247        #take the other positions of the DataFrame, to query those points for
            ↪  erosion process
1248
1249        if np.size(FCC2,0)% stepSize ==0:
1250            folge = np.linspace(stepSize,np.size(FCC2,0),(np.size(FCC2,0)-stepS
                ↪  ize)//stepSize+1)
1251        elif np.size(FCC2,0)<=stepSize:
1252            folge = np.array([np.size(FCC2,0)])
1253        else:
1254            folge = np.zeros((np.size(FCC2,0)-stepSize)//stepSize+2)
1255            folge = np.linspace(stepSize,((np.size(FCC2,0)-stepSize)//stepSize+
                ↪  1)*stepSize,(np.size(FCC2,0)-stepSize)//stepSize+1)
1256            folge[-1] = np.size(FCC2,0)
1257        #subdividing the data point set
1258
1259        targetSize =np.size(positionen[mask,:],0)
1260
1261        del positionen
1262        #free memory
1263        if derode > 0.:
1264        #start of erosion process, derode == 0 skips this step
1265            last = 0
1266            if NN > targetSize:
1267                NN = targetSize
1268            for i in folge:
1269                block = FCC2[last:i,:]
1270                dists = np.zeros((2,2))
1271                while np.min(dists[:,-1]) < derode:
```

```
1272                    #query points as long as last neighbor is smaller than
                        ↪  derode
1273                    results, dists = flannCluster.nn_index(block, NN,
                        ↪  checks=paramsCluster['checks'])
1274                    if np.min(dists[:,-1]) > derode:
1275                        break
1276                    NN+=10
1277                    if NN > targetSize:
1278                        NN = targetSize
1279                results0=np.union1d(results0, results[dists<=derode])
1280                #take unique indices of points which have a distance smaller
                    ↪  than derode
1281                last = i
1282            print(targetSize)
1283            mask2 =  np.ones(targetSize,dtype = bool)
1284            mask2[results0] = False
1285            #create logical array, set to False for found atoms
1286            lClusters = lpos.iloc[mask,:].iloc[mask2,:].copy() #ix replaced
1287            #take only non eroded atoms
1288        else:
1289            lClusters = lpos.iloc[mask,:].copy() #ix replaced
1290            #create lCluster Dataframe of core atoms and linked atoms
1291
1292        flannCluster.delete_index()
1293        #delete kd tree
1294        print('erosion', results0.shape)
1295        #DEBUGGING Info
1296        #-------------------------------------------------------------------
        ↪  ---------- linking together
        ↪  clusters
1297        set_distance_type('euclidean')
1298        flannCluster = FLANN()
1299        #create FLANN Object
1300
1301        clusternumber = np.zeros(np.size(lClusters,0), dtype = np.int32)
1302        #Initializing a array for the cluster number with zero
1303
1304        counterCluster = 1
1305        #initizialize counter for cluster number with 1
1306        paramsCluster = flannCluster.build_index(lClusters.iloc[:,1:4].get_valu
        ↪  es(),algorithm=4, target_precision=1., log_level = "info");
        ↪  #algorithm=4 selects kdtree single #ix ersetzt
1307        #create kdtree of clustered atoms
1308    #-------------------------------------------------------------------
    ↪  ------
    ↪
```

```
1309        FCC =np.array(lClusters.iloc[:,1:4].get_values(), order='C') #ix
            ↪  replaced, FLANN needs the 'C' representation (?)
1310        #create query points
1311    #-------------------------------------------------------------------
        ↪  ------
1312        results0 = np.array([], dtype=np.int32)
1313        for j in FCC:
1314                result = flannCluster.nn_radius(j, dlink,
                        ↪  checks=paramsCluster['checks'])[0]
1315                #query points within radius for one point
1316                #TODO unecessary result and result0
1317                results0 = result
1318                if np.max(clusternumber[results0]) == 0:
1319                    clusternumber[results0] = counterCluster
1320                    counterCluster += 1
1321                    #if maximum cluster number of found point is 0, set
                        ↪  counterumber
1322                    #as cluster number and increase counter
1323                else:
1324                    clusterN = np.min(clusternumber[results0][clusternumber[res
                        ↪  ults0]>0])
1325                    assert clusterN != 0, 'FEHLER'
1326                    #raises error if clusternumber minimum is zero
1327                    for i in np.unique(clusternumber[results0]):
1328                        if i != 0:
1329                            clusternumber[clusternumber == i] = clusterN
1330                        #set every linked together cluster number to the
                            ↪  minimum cluster number
1331                    clusternumber[results0] = clusterN
1332                    #set the minimum cluster number for found within radius
1333        #-------------------------------------------------------------------
            ↪  ----------
1334        print('clusterID')
1335        #DEBUGGING Info
1336        flannCluster.delete_index()
1337        #delete kd tree
1338
1339        tmp = pd.DataFrame({'clN': clusternumber}, index =
            ↪  lClusters.index.get_values())
1340        #create cluster number coloumn pandas DataFrame
1341        lClusters = pd.concat([lClusters,tmp], axis = 1, join_axes=[tmp.index])
1342        #add cluster number to existing DataFrame object for atom probe data
1343
1344        del flannCluster
1345        for i,wert in enumerate(lClusters.clN.unique()):
1346            lClusters.clN.get_values()[lClusters.clN.get_values()==wert]=i+1
```

```
1347        #delete flann Object
1348        return lClusters
1349    def getClusterStat(lClusters, elementCore):
1350        """
1351        creates cluster statistic of core atoms,
1352        histogram of number of atoms of clusters (as size measure)
1353        additional a second histogram is calculated as seen below.
1354        """
1355    #    groessterCluster = lClusters.groupby('clN').clN.count().argmax()
1356    #    kleinsterCluster =
        ↪  lClusters.ix[coreBool].groupby('clN').clN.count().argmin()
1357    #How to find smallest and largest cluster
1358
1359        coreBool = getCoresBool(lClusters, elementCore)
1360        #creates logical array for selected species of clusters (for example
        ↪  Mg, Si atoms)
1361        test = lClusters.ix[coreBool].groupby('clN').clN.count()
1362        #counts the number of entries with same cluster number
1363        H, edges = np.histogram(test, bins=np.arange(np.min(test)-1,
        ↪  np.max(test)+1, 1))
1364        #histograms this information
1365
1366        H2 = np.zeros(np.size(H))
1367        #initializes a second histogram
1368        for i in xrange(np.size(H)):
1369            H2[i] = np.sum(H[i:])
1370            #how many counts are greater than this cluster
1371        return H, H2, edges[:-1]
1372        #returns the histograms, and the left edges (until without last entry
        ↪  [-1])
1373    def plotClusterStat(H, H2, stutz):
1374        """
1375        creates a figure based on the information created in getClusterStat().
1376        """
1377        fig2 = plt.figure()
1378        conv = 2.54
1379        #1 inch = 2.54cm
1380        breite=15/conv
1381        hoehe=10/conv
1382        fig2.set_size_inches(breite,hoehe)
1383        ax2 = fig2.add_subplot(111)
1384
1385        ax2.plot(stutz, H,'-o', label='histogram cluster size')
1386        ax2.plot(stutz, H2, '-x', label='NMin-val')
1387        ax2.set_yscale("log")
1388        #logarithmic (10) y-scale
```

```
1389        ax2.grid(True)
1390        ax2.legend(loc='best')
1391        #adds legend for the labels of the curve, loc=best ... choses best
            ↪  aligment of the legend
1392        return
1393    def summenKurve(a):
1394        rueck = np.zeros(np.size(a))
1395        summe = 0
1396
1397        for i in xrange(np.size(a)):
1398            summe +=a[i]
1399            rueck[i] = summe
1400        #there is a faster way to do this, use np.cumsum()
1401        return rueck
1402    def centerMass0(lposgroup):
1403        """
1404        point center of "mass"
1405        lposgroup ... is a grouped pandas DataFrame (used to calculate center
        ↪  of mass
1406        for a cluster)
1407        """
1408
1409        xmean = lposgroup.x.mean()
1410        ymean = lposgroup.y.mean()
1411        zmean = lposgroup.z.mean()
1412
1413        return np.array([xmean, ymean, zmean])
1414    def radiusGyration(lposgroup):
1415        """
1416        calculates GUINIER radius for grouped pandas DataFrame (used for
        ↪  clusters),
1417        called by getRadiusGyrHist()
1418        """
1419        n = lposgroup.x.count()
1420        #number of atoms used
1421        n = float(n)
1422        #float cast
1423        s = np.array([lposgroup.x.mean(), lposgroup.y.mean(),
            ↪  lposgroup.z.mean()])
1424        #point center of "mass"
1425        rG = np.sqrt(5/3.) * np.sqrt(np.sum(np.power(lposgroup.x-s[0],2)+np.pow
            ↪  er(lposgroup.y-s[1],2)+np.power(lposgroup.z-s[2],2))/n)
1426        #calculated GUINIER radius for grouped values
1427    #    print(rG)
1428        #DEBUGGING Info
1429
```

```
1430     return rG
1431 def getRadiusGyrHist(lpos):
1432     """
1433     GUINIER radius of clusters
1434     """
1435     lposgrouped = lpos.groupby('clN')
1436     #group by cluster number
1437     vals = lposgrouped.apply(radiusGyration)
1438     #applies a function to grouped values
1439     H, edges = np.histogram(vals, bins = 100)
1440     #histograms values
1441     radGyr_mean = np.mean(vals)
1442     #mean GUINIER radius value
1443     stutz = (edges[1:]+edges[:-1])/2.
1444 
1445     return H, stutz, radGyr_mean
1446 def getCompCluster(lClusters, listElement):
1447     """
1448     should calculate compositions of clusters,
1449         is called by plotClusterComp()
1450     """
1451 
1452     binsComp = np.linspace(0,1,200)
1453     #create bin array
1454     clusterAnzahl = lClusters.clN.max()
1455     #assumes that cluster number is equal to count of clusters,
1456     #better would be lpos.groupby('clN').comp.count().count(), fixed:
1457     #getClusterAtomsL() is modified
1458     list0 = []
1459         #Initialize list to return
1460 
1461     index1 = (lClusters.groupby(['clN', 'comp']).comp.count()/lClusters.gro
     ↪  upby(['clN']).comp.count()).index.get_level_values(1)
1462     #gets index for species of elements of the grouped
     ↪  lClusters.groupby(['clN', 'comp'])
1463     for i in listElement:
1464         index2 = index1 == i
1465         H, edges = np.histogram((lClusters.groupby(['clN',
         ↪  'comp']).comp.count()/lClusters.groupby(['clN']).comp.count())[
         ↪  index2].get_values(),
         ↪  bins=binsComp)
1466                 #histograms the fraction for entry in listElement
1467         anzahlClusterEnt = (lClusters.groupby(['clN', 'comp']).comp.count()
         ↪  /lClusters.groupby(['clN']).comp.count())[index2].count()
1468                 #counts the number of clusters which do contain the entry
                     ↪  in listElement
```

```
1469            H[0] += clusterAnzahl - anzahlClusterEnt
1470
1471                    #sets the zero entry, which is otherwise not counted due
                        ↪  to the way the pandas arithmentics work
1472            stutz = (edges[1:]+edges[:-1])/2.
1473
1474            mean = (lClusters.groupby(['clN', 'comp']).comp.count()/lClusters.g
                ↪  roupby(['clN']).comp.count())[index2].sum() /
                ↪  anzahlClusterEnt
1475                    #mean without zero content
1476            list0.append([H, stutz, i, mean])
1477
1478    return list0
1479 def plotClusterComp(lClusters, listElement, titel, max0 = 0.15, ):
1480    fig = plt.figure()
1481    conv = 2.54
1482 #    fig.set_size_inches(15/conv, 7.5/conv)
1483    fig.set_size_inches(10, 10)
1484    ax = fig.add_subplot(111)
1485        #create figure and add subplot
1486
1487 #    listElement = lClusters.groupby('comp').count().index
1488 #    listElement = ['Si:1', 'Mg:1', 'Mn:1', 'Cu:1', 'Sn:1', 'Ga:1']
1489
1490    listClusterComp = getCompCluster(lClusters, listElement)
1491        #get statistics of Clusters
1492
1493    max0 = 0.
1494    map0 = cm.nipy_spectral
1495        #choose a colourmap
1496    counter = 0
1497    n = float(len(listClusterComp))
1498        #number of entries listClusterComp as float
1499
1500    for i in listClusterComp:
1501        ax.plot(i[1], i[0], label=i[2], lw=1.5, c = map0(counter/n))
1502                #plots the histograms for each element, with a differing
                    ↪  colour
1503        ax.axvline(i[3], c = map0(counter/n), lw = 1.5)
1504                #adds vertical lines for the mean (without zero values)
1505        if i[1][i[0]>0][-1] > max0:
1506                #maximum x value for y values which are > 0
1507            max0 = i[1][i[0]>0][-1]
1508                        #sets new max0 for x axis adjustment
1509            ax.set_xbound(0., max0)
1510        counter += 1
```

```
1511        ax.set_xbound(0., max0)
1512            #adjusts x axis
1513
1514        ax.legend(fontsize = 6)
1515        ax.grid(True)
1516        ax.set_ylabel('number clusters')
1517        ax.set_xlabel('concentration')
1518        ax.set_title(titel)
1519            #sets legend, grid, x,y axis labeling and title
1520        return fig, ax
1521    def plotClusterSize(lClusters, titel):
1522        fig2 = plt.figure()
1523        fig2.set_size_inches(15.,10.)
1524        ax = fig2.add_subplot(111)
1525            #creates figure and adds subplot
1526
1527        H, stutz, radGyr = getRadiusGyrHist(lClusters)
1528            #gets Guinier radius statistics
1529
1530        ax.plot(stutz, H, '-o')
1531        ax.legend()
1532        ax.grid(True)
1533        ax.set_ylabel('counts')
1534        ax.set_xlabel('radius')
1535        ax.set_title(titel)
1536            #plots statistics, sets grid, labels x,y, title
1537        return fig2
1538    def getCompPos(pos, rrngs, path, deltaDalton = 0.02, nrUpperboundFit =
    ↪  1000, startVal0 = 1., startVal1 = 10.):
1539        """
1540        creates figure of mass-to-charge ratio spectrum of pos file, fits
    ↪  background and plots
1541        resiudal spectrum, saves calculated composition to path
1542        """
1543        print(path)
1544            #DEBUGGING Info
1545
1546        fig = plt.figure()
1547        conv = 2.54
1548        #1 inch = 2.54cm
1549        breite=15/conv
1550        hoehe=2*10/conv
1551        fig.set_size_inches(breite,hoehe)
1552        ax = fig.add_subplot(211)
1553        ax2 = fig.add_subplot(212)
1554        ax.grid(True)
```

```
1555        #creates a figure and adds two subplots
1556
1557        lpos = label_ions(pos, rrngs)
1558        nBins = np.int32((np.max(pos.Da)-np.min(pos.Da))/deltaDalton)
1559    #    print(nBins)
1560        #DEBUGGING Info
1561
1562        H, edges = np.histogram(pos.Da, nBins)
1563        #histogram of the mass-to-charge ratio of atoms in pos file
1564        H2 , edges2 = np.histogram(lpos.ix[lpos.comp == ''].Da, nBins)
1565        #histogram of the mass-to-charge ratio for unranged ions
1566
1567        stutz = (edges[1:] + edges[:-1])/2.
1568        stutz2 = (edges2[1:] + edges2[:-1])/2.
1569
1570        comp = []
1571        name = []
1572
1573        popt, pcov = curve_fit(func1, stutz2[H2>0] , H2[H2>0], p0=(startVal0,
            ↪  startVal1))
1574        #curve fitting of func1 to the unranged points in the histograms
1575        cut = np.ceil(func1(stutz, *popt))
1576        #rounds function values, to avoid non-integer counts for a histogram
1577
1578        ax.plot(stutz, H,'-', lw = 0.5)
1579        ax.plot(stutz, cut,'-')
1580        #plots mass-to-charge state and background fit
1581
1582        ax.set_title(path.split('/')[5] + ' background fit')
1583        ax.set_xlabel('mass-to-charge [Da]')
1584        ax.set_ylabel('counts')
1585        ax.set_yscale('log')
1586        #labeling, log scale setting for first subplot
1587
1588        ax2.set_title('residual spectrum')
1589        ax2.set_xlabel('mass-to-charge [Da]')
1590        ax2.set_ylabel('counts')
1591        ax2.set_yscale('log')
1592        #labeling, log scale setting for second subplot
1593
1594        rrngs = rrngs.sort('lower')
1595        for i, wert in enumerate(rrngs.lower):
1596            ax2.axvline(wert, color='black', lw = 0.1)
1597            ax2.axvline(rrngs.upper[i], color='r', lw = 0.1)
1598            #draw vertical lines for upper and lower bound of range
```

```
1599            ax2.annotate(rrngs.comp[i].replace('Name:',''), xy=(wert, 100. +
                ↪  (i%4)*10**(i%4+1)), fontsize = 4)
1600            #draw name of ion at x = lower range, y = 100, 2000, 30000 ...
                ↪  (logscale)
1601
1602        H = H - cut
1603        #residual histogram
1604        H[H<0] = 0
1605        #filters zero counts
1606
1607        ax2.plot(stutz[H>0], H[H>0],'-', lw = 0.5)
1608        ax2.set_xbound(0,75)
1609        ax2.grid(True)
1610        #plots residual histogram
1611
1612        df2 = pd.DataFrame({'counts' : H,
1613                            'edges' : stutz})
1614
1615        for g,d in rrngs.groupby('comp'):
1616            count = 0
1617            for i in d.iterrows():
1618                count += df2.ix[(df2.edges >= i[1].lower) & (df2.edges <=
                    ↪  i[1].upper)].counts.sum()
1619
1620            comp.append(count)
1621            name.append(g)
1622        #count composition of residuals
1623
1624        comp = np.array(comp)/np.float(np.sum(comp))
1625        output = ""
1626        output += 'ion %\n'
1627        print('ion %')
1628        for i in range(len(name)):
1629            print(name[i] +" "+ str(100.*comp[i]))
1630            output += name[i] + " " + str(100.*comp[i]) + "\n"
1631        #creates string for txt output
1632
1633        fig.tight_layout()
1634        f = open(path,'w')
1635        f.write(output)
1636        f.close()
1637        #saves txt file to path
1638        return fig
1639    def func1(m, a, b):
1640        """fitting function for background fitting, called by getCompPos()"""
1641        y = a/(np.sqrt(m)+b)
```

```
1642        return y
1643    def hitlabel(d):
1644        """
1645        adds a coloumn which contains the .count() of epos.id
1646        """
1647        d.loc[:,'hn'] = d.id.count()
1648        return d
1649    def getStandardFig():
1650        """
1651        returns a figure in  a size which is often used in apt_importers
1652        """
1653        fig2 = plt.figure()
1654        conv = 2.54
1655        #1 inch = 2.54cm
1656        breite=15/conv
1657        hoehe=7.5/conv
1658        fig2.set_size_inches(breite,hoehe)
1659        ax2 = fig2.add_subplot(111)
1660        return fig2, ax2
1661    def deleteBG(z):
1662        """
1663        used to delete the background from a z SDM, used should be the value
    ↪   of
1664        the first minimum of the z SDM as a stopping criteria for smoothing
1665        """
1666        firstMin = findFirstMin(z)
1667        z = z.copy()
1668        z00 = z.copy()
1669        #create two copies of the z array
1670        while(firstMin < np.max(z)):
1671            z0 = z.copy()
1672            for i in xrange(1, np.size(z)-1):
1673                z[i] = min(z0[i], (z0[i+1]+z0[i-1])/2.)
1674                #smooth function until maximum is smaller than original first
                    ↪   minium
1675        z = z00 - z
1676        #delete background (smoothed function)
1677        z = z / np.float(np.sum(z))
1678        #norm to all counts
1679        return z
1680    def findFirstMin(z):
1681        """is called by deleteBG(), w_zSDM_Al_Cluster has to be in the memory
        ↪   known,
1682         to work with this function"""
1683        i0 = z.argmax()
1684        #index of z maximum
```

```
1685        i1 = i0+1
1686        i2 = i0+2
1687        isearch = i0
1688        while i0 < np.size(z)-3:
1689            #avoid index errors
1690            if z[i0] < z[i1] and z[i0] < z[i2]:
1691                isearch = i0
1692                break
1693            else:
1694                i0 += 1
1695                i1 += 1
1696                i2 += 1
1697                #increment counting integers
1698        #searches a local minimum in the z array, and returns the first
            ↪  minimum value
1699        return z[isearch]
1700    def norm(z):
1701        """norms to the sum of counts"""
1702        z = z / np.float(np.sum(z))
1703        return z
1704    def normsmooth(z, N):
1705        """returns difference to the smoothed function, norms it to the sum
            ↪  of all counts"""
1706        z = z-smoothingZSDM(z,N)
1707        z = z / np.float(np.sum(z))
1708        return z
1709    def richtungsWinkel(v1,v2):
1710        """returns angle between vectors for normed vectors v1, v2"""
1711        phi = np.arccos(np.dot(v1,v2))*180./np.pi
1712        return phi
1713    def findMaxAll(z):
1714        """
1715        should find all positive maxima of z SDM
1716        """
1717        listMax = []
1718        i1 = z.argmax()
1719        i0 = i1-2
1720        i2 = i1+2
1721
1722        while i0 < np.size(z)-4:
1723            if (z[i1] > z[i2] and z[i1] > z[i0] and z[i1] > z[i2-1] and z[i1] >
                ↪  z[i0+1]):
1724                listMax.append(i1)
1725    #         print(i0,i1,i2,z[i1] < z[i2],z[i1] > z[i0])
1726    #                 DEBGUGGING Info
1727            i0 += 1
```

```
1728            i1 += 1
1729            i2 += 1
1730        return listMax
1731    def angleMiller(a,b):
1732        """calculates the angle in between two vectors in degree"""
1733        a=a/scipy.linalg.norm(a)
1734        b=b/scipy.linalg.norm(b)
1735        phi=np.arccos(np.dot(a,b))*180./np.pi
1736        return phi
1737    def zoneMiller(a,b):
1738        """calculates the zone axis of given directions, returns a normed
            ↪ vector"""
1739        a=a/scipy.linalg.norm(a)
1740        b=b/scipy.linalg.norm(b)
1741        c = np.cross(a,b)
1742        return c
1743    def colorClusters(lClusters):
1744        """colours clusters for better visualization"""
1745        colorList = []
1746        colorList.append('#FF00FF')
1747        colorList.append('#FFFF00')
1748        colorList.append('#FFC800')
1749        colorList.append('#FFAFAF')
1750        colorList.append('#FF0000')
1751        colorList.append('#00FF00')
1752        colorList.append('#0000FF')
1753        colorList.append('#00FFFF')
1754            #creates a list of hex colour values
1755        i = 0
1756        for x in lClusters.clN.unique():
1757            #for every cluster
1758            lClusters.loc[lClusters.clN == x, ['colour']] = colorList[i]
1759                    #colours a cluster
1760            i +=1
1761                    #increases counting integer
1762            if i > np.size(colorList)-1:
1763                    #sets counting integer to zero if counter is through
                        ↪ colorList
1764                i=0
1765        return
1766    def rrngsToRng(rrngs, ions):
1767        """converts the info read by a .rrng range file to a rng file format
1768        and prints it to the output"""
1769        print(str(ions.shape[0])+" "+str(rrngs.shape[0]))
1770            #number of ions and number of ranges
1771        for i in ions.name:
```

```
1772            print(i)
1773                    #names of ions
1774            if rrngs[np.logical_not(rrngs.comp.str.contains(' ')) &
                ↪  rrngs.comp.str.contains(i)].size >0:
1775                values = hex2rgb(rrngs[np.logical_not(rrngs.comp.str.contains('
                    ↪  ')) & rrngs.comp.str.contains(i)].colour.unique()[0])
1776                    #saves colourvalues for the not complex ion
1777            else:
1778                values = hex2rgb(rrngs[rrngs.comp.str.contains(i)].colour.uniqu⌋
                    ↪  e()[0])
1779            print(", ".join(['%.2f' %j for j in values]))
1780                    #prints rgb color with 2 digit precision separated by ", "
1781
1782        print('----------------- ' + " ".join(ions.name))
1783            #prints '----------------- ' and ion names separated by spaces
1784        for n,r in rrngs.iterrows():
1785    #         print(n)
1786    #                 DEBUGGING Info
1787            listCount = np.zeros(ions.shape[0],dtype=int)
1788                    #Initialize integer array for every loop
1789            for i in r.comp.split(" "):
1790    #             print(i.split(':')[0])
1791    #                         DEBUGGING Info
1792                if(i.split(':')[0] != 'Name'):
1793                            #if entry is not a non-identified ion
1794                    listCount[(ions.name == i.split(':')[0]).get_values()] =
                        ↪  int(i.split(':')[1])
1795                                    #set number of element to the entry, e.g.
                                        ↪  Al:2 H:1,
1796                                    #sets at the index of Al in ions, 2; at
                                        ↪  the position of H, 1.
1797                else:
1798                            #if entry is a non-identified ion, starting with
                            ↪  identifier 'Name:' (IVAS default)
1799                    listCount[(ions.name == i.split(':')[1]).get_values()] = 1
1800                                    #set the certain array entry to one
1801            print('. ' + '%.4f' %r.lower +" "+'%.4f' %r.upper+" "+"
            ↪  ".join(['%i' %j for j in listCount]))
1802                    #prints '. ' and with 4 digit precision the upper and
1803                    #lower bound of the range for the given ion.
1804        print("\n")
1805        print('--- polyatomic extension')
1806        return
1807    def hex2rgb(hex_str):
1808        """
1809        hex value (as string) to rgb value conversion,
```

```
1810        called by rrngsToRng()
1811        """
1812        r, g, b = hex_str[:2], hex_str[2:4], hex_str[4:]
1813        rgb = [int(n, base=16)/255. for n in (r, g, b)]
1814        return rgb
1815    def searchMax(data, psibereich, thetabereich):
1816        """
1817        searches psi and theta (rotation angles) for maximum z SDM value
1818        data ... array of difference vectors
1819        psibereich ... array for psi search space
1820        thetabereich ... array for psi search space
1821        """
1822        zmaxbereich = 2. #plus minus delta z space
1823        aufl = 0.005          #resolution for the histogram in nm
1824        zmax = 0                 #variable for the z SDM maximum value
1825        psi0 = 0.                 #value to hold psi position of maximum
1826        theta0 = 0.                 #value to hold theta position of maximum
1827        raster = np.zeros((np.size(psibereich),np.size(thetabereich)))
1828            #two dimensional array for holding the z SDM maximum values
1829        for i,wert in enumerate(psibereich):
1830            for j,wert2 in enumerate(thetabereich):
1831                x,z = getRotateZSDM(zmaxbereich,data,wert,wert2,2*zmaxbereich/a
                    ↪ ufl)
1832                raster[i,j] = np.max(z)
1833                            #saves max z SDM value to 2D array
1834                if np.max(z) > zmax:
1835                    psi0 = wert
1836                    theta0 = wert2
1837                    zmax = np.max(z)
1838                                    #if z SDM max is greater than current max
1839                                    #save current psi and theta value
1840
1841        return raster, psi0, theta
1842    def calculateMginClusters(group, Nmin=0):
1843        countMg = group[group.comp == 'Mg:1'].comp.count()
1844        countSi = group[group.comp == 'Si:1'].comp.count()
1845    #    print(countMg)
1846    #    print(countSi)
1847        summeMgSi = countMg+countSi
1848        if summeMgSi < Nmin:
1849            return np.NaN
1850        else:
1851            return np.float(countMg)/summeMgSi
1852    def calculateMginClustersall(group, Nmin=0):
1853        countMg = group[group.comp == 'Mg:1'].comp.count()
1854        countSi = group[group.comp == 'Si:1'].comp.count()
```

```
1855     summeMgSi = countMg+countSi
1856     if summeMgSi < Nmin:
1857         return np.NaN
1858     else:
1859         return countMg
```

8.3 plot_multiple_hits_Si.py

```
1860 from pyflann import *
1861 #from numpy import *
1862 import time
1863 from apt_importers import *
1864 #import scipy.linalg as lin
1865 from Vis import volvis
1866 from matplotlib import cm
1867 from scipy.special import gamma
1868 from scipy.misc import factorial
1869 import itertools as it
1870 from numpy import linalg as LA
1871 #imports several libraries also the self-written/enhanced apt_importers
1872 """
1873 this script is used to create detectorhitmaps for artifact detection, and
     ↪ visual
1874 inspection. Also code for multiple hits analysis contained
1875 """
1876 def getSiSi(d):
1877     """returns Si doubles"""
1878     if d.loc[d.comp =='Si:1'].comp.count() == 2:
1879         r = d.loc[d.comp =='Si:1']
1880         return r
1881     return
1882 def printrows(d):
1883     """
1884     creates entries True if more than two atoms are in grouped
1885     """
1886     if d.comp.count() >= 2:
1887         d.loc[:,'in'] = True
1888     else:
1889         d.loc[:,'in'] = False
1890     return d
1891 def getDist(d):
1892     """
1893     calculates distances for every combination to first multiple hit ion
1894     """
1895     det_x, det_y = d.det_x.get_values(), d.det_y.get_values()
1896     a = np.array([det_x[0], det_y[0]])
```

```
1897     minDist = 15.
1898     for i in it.combinations(np.arange(0,np.size(det_x),1),2):
1899         b = np.array([det_x[i[0]], det_y[i[0]]])
1900         a = np.array([det_x[i[1]], det_y[i[1]]])
1901         dist = LA.norm(b-a,2)
1902         if LA.norm(b-a,2) < minDist:
1903             minDist = dist
1904 
1905     d.loc[:,'dist'] = minDist
1906     return d
1907 #definition of analysis/test functions
1908 
1909 numberPlots = 4
1910 fig, axes = plt.subplots(1,numberPlots, figsize = (20,5))
1911 axes = axes.flatten()
1912 #create figure and subplotts and flattens array to adress it as one
    ↪  dimensional
1913 #array
1914 
1915 workingDir = '/media/phillip/Volume1/DatenIVAS/'
1916 
1917 
1918 
1919 path = '/media/phillip/Volume1/Messungen_Zuerich_Sep_2018/R34_05976/recons/
    ↪  recon-v01/default/R34_05976-v01.epos'
1920 pathRange = '/media/phillip/Volume1/13_rangeFiles/R34_05935_AlOH_V.rrng'
1921 
1922 epos = read_epos(path)
1923 #loads epos file
1924 
1925 ions,rrngs = read_rrng(pathRange)
1926 #loads rrng file
1927 epos = label_ions(epos, rrngs)
1928 #creates ranged epos file
1929 
1930 elementCore = []
1931 elementCore.append('Mg:1')
1932 elementCore.append('Si:1')
1933 #creates a list and entries 'Mg:1' and 'Si:1'
1934 coreBool = getCoresBool(epos, elementCore)
1935 #creates logical array of positions core Elements
1936 
1937 #-----------------------------------------------------------------------
1938 #multiples = epos[epos.ipp !=1]
1939 #gets ions for which ions per pulse is not 1, i.e. all multiple hit ions
1940 #
```

```
1941 #multiples.loc[:,'id'] = multiples.ipp.cumsum()
1942 #gives multiple ions same "id"
1943 ##multiples = multiples.loc[:,['x','y','z', 'comp', 'det_x', 'det_y',
     ↪  'id']]
1944 #multipleSi = multiples.ix[multiples.comp == 'Si:1',:]
1945 #multiple Si hits
1946 #
1947 #rueck = multipleSi.groupby('id').apply(printrows).loc[:,'in']
1948 #multipleSi = multipleSi.loc[rueck,:]
1949 #multipleSi = multipleSi.groupby('id').apply(getDist)
1950 #multipleSi = multipleSi.ix[multipleSi.dist < 1.]
1951 #selects Si multiple ions which are separated by less than 1 mm
1952 #
1953 #ax.clear()
1954 #ax.hist2d(epos.det_x.get_values(), epos.det_y.get_values(), bins = 100)
1955 #ax.scatter(multipleSi.det_x.get_values(),
     ↪  multipleSi.det_y.get_values(),s = 2, c = 'black')
1956 #ax.set_title('Si-Si hits, detector position')
1957 #fig.savefig(workingDir+'SiSi_multipleHits_detectorSpace.png', dpi = 300)
1958 #plots the positions on the detector as scatter and saves the figure
1959
1960 #multipleSi = multiples.loc[(multiples.comp == 'Ga:1')]
1961 #ax.clear()
1962 #ax.hist2d(epos.det_x.get_values(), epos.det_y.get_values(), bins = 100)
1963 #ax.scatter(multipleSi.det_x.get_values(),
     ↪  multipleSi.det_y.get_values(),s = 2, c = 'gray')
1964 #ax.set_title('Ga-multiple hits, detector position')
1965 #fig.savefig(workingDir+'Ga_multipleHits_detectorSpace.png', dpi = 300)
1966 #plots Ga multiple hits
1967
1968 #multipleSi = multiples.loc[(multiples.comp == 'Si:1')]
1969 #ax.clear()
1970 #ax.hist2d(epos.det_x.get_values(), epos.det_y.get_values(), bins = 100)
1971 #ax.scatter(multipleSi.det_x.get_values(),
     ↪  multipleSi.det_y.get_values(),s = 2, c = 'gray')
1972 #ax.set_title('Si-multiple hits, detector position')
1973 #plots Si multiple hits
1974 #-----------------------------------------------------------------------
1975 #multiple hits analysis
1976
1977 axes[0].hist2d(epos.det_x.get_values(), epos.det_y.get_values(), bins = 150)
1978 #plots 2d histogram of detector coordinates of all detected atoms
1979 axes[0].set_title('detector hitmap', fontsize = 20)
1980 axes[1].hist2d(epos.ix[epos.comp == 'Si:1'].det_x.get_values(),
     ↪  epos.ix[epos.comp == 'Si:1'].det_y.get_values(), bins = 80)
```

```
1981 #axes[1].hist2d(epos.ix[epos.comp == 'Zn:1'].det_x.get_values(),
     ↪  epos.ix[epos.comp == 'Zn:1'].det_y.get_values(), bins = 80)
1982 axes[1].set_title('Si', fontsize = 20)
1983 #axes[1].set_title('Zn', fontsize = 20)
1984 #axes[1].hist2d(epos.loc[epos.comp == 'Sc:1'].det_x.get_values(),
     ↪  epos.loc[epos.comp == 'Sc:1'].det_y.get_values(), bins = 80)
1985 #axes[1].set_title('Si', fontsize = 20)
1986 axes[2].hist2d(epos.loc[epos.comp == 'Cu:1'].det_x.get_values(),
     ↪  epos.loc[epos.comp == 'Cu:1'].det_y.get_values(), bins = 80)
1987 axes[2].set_title('Cu', fontsize = 20)
1988 #axes[2].hist2d(epos.loc[epos.comp == 'Ga:1'].det_x.get_values(),
     ↪  epos.loc[epos.comp == 'Ga:1'].det_y.get_values(), bins = 80)
1989 #axes[2].set_title('Ga', fontsize = 20)
1990 #axes[2].hist2d(epos.loc[epos.comp == 'Mn:1'].det_x.get_values(),
     ↪  epos.loc[epos.comp == 'Mn:1'].det_y.get_values(), bins = 80)
1991 #axes[2].set_title('Mn', fontsize = 20)
1992 axes[3].hist2d(epos.loc[epos.comp == 'Mg:1'].det_x.get_values(),
     ↪  epos.loc[epos.comp == 'Mg:1'].det_y.get_values(), bins = 80)
1993 axes[3].set_title('Mg', fontsize = 20)
1994 #plots 2d histogram of detector coordinates of specific species
1995
1996 #coreShow = getCoresBool(epos, ['Si:1', 'Ga:1'])
1997 #volvis(epos[coreShow])
1998 #to look at the Si and Ga atoms in 3D
1999
2000 #saveDir = '/media/phillip/Volume1/Daten_PA/R21_08675/auswertung/'
2001 #
2002 [ax.set_xlabel('det x / mm', fontsize = 20) for ax in axes]
2003 [ax.set_ylabel('det y / mm', fontsize = 20) for ax in axes]
2004 #labels alle axes in figure
2005
2006 ##ax.set_title('Si', fontsize = 20)
2007 ##fig.savefig('/media/phillip/Volume1/Dropbox/poster_MSE/figure_new/AQ_hi
     ↪  tmap.tiff', bbox_inches = 'tight', dpi =
     ↪  300)
2008 ##fig.savefig(saveDir + 'Si_detHitmap', bbox_inches = 'tight', dpi = 300)
2009 #fig.savefig(saveDir + 'Cu_detHitmap', bbox_inches = 'tight', dpi = 300)
2010 #fig.savefig('/media/phillip/Volume1/DatenIVAS/R34_04367/auswertung/detHi
     ↪  tmap_04367.png', bbox_inches = 'tight', dpi =
     ↪  300)
2011
2012 fig.tight_layout()
2013 #hinders that axis text overwrites lines of diagram
2014
```

```
2015 fig.savefig('/'.join(path.split('/')[:-4])+ '/auswertung/'+
     ↪  path.split('/')[-1][:-5] + '_detectorHitmaps.png', bbox_inches =
     ↪  'tight', dpi = 300)
2016 #used to save the created figure into analysis directory
2017
2018 #deltaDalton = 0.02
2019 #nBins = np.int((np.max(epos.Da)-np.min(epos.Da))/deltaDalton)
2020 #H, edges = np.histogram(epos.Da, nBins)
2021 #constCutoff = 0
2022 #xax = (edges[1:]+edges[:-1])/2.
2023 #axes[2].plot(xax[H>constCutoff],H[H>constCutoff],'.-')
2024 #axes[2].set_yscale("log")
2025 #for i, wert in enumerate(rrngs.lower):
2026 #    axes[2].axvline(wert, color='black')
2027 #    axes[2].axvline(rrngs.upper[i], color='r')
2028 #axes[2].axhline(100)
2029 #axes[2].set_xbound(0,70)
2030 #plots mass-to-charge ratio and the ranges of elements in rrng
2031
2032 #[ax.set_xbound(-15,15) for ax in axes]
2033 #[ax.set_ybound(-15,15) for ax in axes]
2034
2035 #p1=np.array([0.87,0.85])
2036 #p2=np.array([6,-5.])
2037 #k = (p2[1]-p1[1])/(p2[0]-p1[0])
2038 #x = np.linspace(-15,15,100)
2039 ##p2new = [3,3]
2040 #p2new = [-2,-2]
2041 #y= k*(x-p2new[0])+p2new[1]
2042 #axes[1].plot(x,y,c='k',lw=1.5)
2043 #axes[0].plot(x,y,c='k',lw=1.5)
2044 #p2new = [3,3]
2045 #p2new = [-2,-2]
2046 #y= k*(x-p2new[0])+p2new[1]
2047 #axes[1].plot(x,y,c='k',lw=1.5)
2048 #axes[0].plot(x,y,c='k',lw=1.5)
2049 #fig.savefig('/'.join(path.split('/')[:-4])+ '/auswertung/'+
     ↪  path.split('/')[-1][:-5] + '_detectorHitmaps_cut.png', bbox_inches =
     ↪  'tight', dpi = 300)
2050
2051 #cut1 = epos[epos.det_y > k*(epos.det_x-3)+3]
2052 #cut2 = epos[epos.det_y < k*(epos.det_x+2)-2]
2053 #
2054 #cut = cut1.append(cut2)
```

8.4 analyse_recon.py

```
2055 from pyflann import *
2056 #from numpy import *
2057 import time
2058 from apt_importers import *
2059 #import scipy.linalg as lin
2060 from Vis import volvis
2061 from matplotlib import cm
2062 from scipy.special import gamma
2063 from scipy.misc import factorial
2064 import itertools as it
2065 from numpy import linalg as LA
2066 from scipy.optimize import curve_fit
2067 from scipy import interpolate
2068
2069 def fitX((x,y),*a):
2070     p = a[0]+a[1]*y+a[2]*x+a[3]*np.power(x,2)+a[4]*np.power(y,2)+a[5]*np.po
      ↪  wer(x,3)+a[6]*x*np.power(y,2)+a[7]*np.power(x,4)+a[8]*np.power(x,3)
      ↪  *y+a[9]*np.power(x,2)*np.power(y,2)+a[10]*x*np.power(y,3)+a[11]*np.
      ↪  power(y,4)
2071     return p
2072 def fitX2((x,y),*a):
2073     p = a[0]+a[1]*np.power(x,2)+a[2]*np.power(y,2)+a[3]*np.power(x,4)+a[4]*
      ↪  np.power(x,2)*np.power(y,2)+a[5]*np.power(y,4)
2074     return p
2075 def fitX3((x,y),*a):
2076     p = a[0]+a[1]*x+a[3]*np.power(y,2)+a[4]*np.power(x,3)+a[5]*x*np.power(y
      ↪  ,2)+a[6]*np.power(x,2)*np.power(y,2)
2077     return p
2078 def fitY((x,y),*a):
2079     p = a[0]+a[1]*y+a[2]*x+a[3]*x*y+a[4]*np.power(y,2)+a[5]*np.power(x,2)*y
      ↪  +a[6]*np.power(y,3)+a[7]*np.power(x,3)*y+a[8]*x*np.power(y,3)
2080     return p
2081 def FitEntlangY(x,*a):
2082     p = a[0]+a[1]*x+a[2]*np.power(x,2)
2083     return p
2084 def FitEntlangY2(x,*a):
2085     p = a[0]+a[1]*np.power(x,2)+a[2]*np.power(x,4)
2086     #looks good, fit along Y für die det_x-det_x2
2087     return p
2088 def FitEntlangY3(x,*a):
2089     p = a[0]+a[1]*x+a[2]*np.power(x,2)+a[3]*np.power(x,3)+a[4]*np.power(x,4
      ↪  )+a[5]*np.power(x,5)
2090     #looks good, fit along X für die det_x-det_x2
2091     return p
```

```
2092 def FitXYlin((x,y),*a):
2093 #     x = x-a[11]
2094 #     y = y-a[12]
2095     p = a[0]+a[1]*np.power(y,2)+a[2]*np.power(y,4)+a[3]*x+a[4]*np.power(x,2
     ↪  )+a[5]*np.power(x,3)\
2096     +a[6]*np.power(x,4)+a[7]*np.power(x,5)+a[8]*x*np.power(y,2)+a[9]*x*np.p
     ↪  ower(y,4)\
2097     +a[10]*np.power(x,2)*np.power(y,2)+a[11]*np.power(x,3)*np.power(y,2)
2098 
2099     return p
2100 def FitXYlin_test((x,y),*a):
2101 #     x = x-a[11]
2102 #     y = y-a[12]
2103     p = a[0]+a[1]*np.power(y,2)+a[2]*np.power(y,4)+a[3]*x+a[4]*np.power(x,2
     ↪  )+a[5]*np.power(x,3)\
2104     +a[6]*np.power(x,4)+a[7]*np.power(x,5)+a[8]*x*np.power(y,2)+a[9]*x*np.p
     ↪  ower(y,4)\
2105     +a[10]*np.power(x,2)*np.power(y,2)
2106 
2107     return p
2108 def fit_y2(x,*a):
2109     p = a[0]+a[1]*x+a[2]*np.power(x,2)+a[3]*np.power(x,3)
2110     return p
2111 def fit_y2_2(x,*a):
2112     p = a[0]+a[1]*x+a[2]*np.power(x,2)
2113     return p
2114 def FitXYlin2((x,y),*a):
2115     x = x-a[0]
2116     y = y-a[1]
2117     p = a[2]+a[3]*x+a[5]*y+a[6]*np.power(y,2)+a[7]*np.power(y,3)+a[4]*np.po
     ↪  wer(x,2)\
2118     +a[9]*y*np.power(x,2)+a[8]*y*x
2119     return p
2120 def fitVoltage(x,*a):
2121     p = a[0]+a[1]*np.sqrt(x)+a[2]*x
2122     return p
2123 def calcDetkorr(x2, poptX, poptY):
2124     x2_new = (x2[0]-FitXYlin(x2,*poptX), x2[1]-FitXYlin2(x2,*poptY))
2125     return x2_new
2126 #functions which were tried to find a connection between recalculated
     ↪  detector
2127 #coordinates and original detector coordinates
2128 
2129 """
2130 script calculates some approximate virtual flight path for reconstruction
2131 calibration, builds new reconstruction upon an existing one.
```

```
2132 """
2133
2134 numberPlots = 4
2135 fig, axes = plt.subplots(1,numberPlots, figsize = (20,5))
2136 axes = axes.flatten()
2137 #create figure and make array of subplots one dimensional
2138 path = '/media/phillip/Volume1/Daten7050/R21_09743/recons/recon-v01/default
     ↪  /R21_09743-v01.epos'
2139 ICF = 1.225
2140 kf = 4.54
2141 #paths and corresponding used kf and ICF values
2142
2143 epos = read_epos(path)
2144 #load epos file
2145
2146 pathRange = '/media/phillip/Volume1/13_rangeFiles/R21_09743_v01.rrng'
2147 #path for range file
2148
2149 ions,rrngs = read_rrng(pathRange)
2150 #load range(rrng) file
2151 #epos = epos.loc[:,['x', 'y', 'z', 'Da', 'det_x', 'det_y', 'comp', 'ipp']]
2152 epos = label_ions(epos, rrngs)
2153 #label ranged ions
2154
2155 epos = epos.loc[:,['x','y','z','det_x','det_y','det_x2','det_y2','DC_kV','R
     ↪  ','Da']]
2156 #reduce epos file and add det_x2, det_y2 and R coloumn
2157 #kf = 4.4
2158 #ICF = 1.201
2159
2160 #kf = 4.6
2161 #ICF = 1.2
2162
2163 #kf = 3.6
2164 #ICF = 1.32
2165
2166
2167 FevAl = 19.
2168 #evaporation field in V/nm
2169 initial = np.ones((12))
2170 #start values for a fit
2171
2172 #epos = epos[:8*10**6]
2173 #epos = epos.loc[epos.det_x**2 + epos.det_y**2 <15.**2]
2174 #cuts of the whole datasets
2175
```

```
2176 epos.R = 1.2*epos.DC_kV/(FevAl*kf)
2177 #calculation of radius via using the voltage, 1.2 due to 20% pulse
     ↪ fraction
2178 #epos.R = epos.DC_kV/(FevAl*kf)
2179
2180 #poptV, pcovV = curve_fit(fitVoltage, epos[::1000].index.get_values(),
     ↪ epos[::1000].R, p0 = initial)
2181 #fitting
2182 epos = epos.loc[:,['x','y','z', 'det_x','det_y','det_x2','det_y2','R','Da'
     ↪ ]]
2183 #delete DC_kV coloumn
2184
2185 #epos.R = fitVoltage(epos.index.get_values(), *poptV)
2186 #epos.iloc[(epos.det_y/epos.y).argmin()]
2187
2188 dety0 = epos.ix[(epos.det_y/epos.y).argmin()].det_y
2189 detx0 = epos.ix[(epos.det_x/epos.x).argmin()].det_x
2190 m = ICF-1.
2191 L = 382.
2192 #standard flight distance printed in IVAS
2193
2194 epos.det_x2 = epos.x/np.sqrt(np.power(epos.x,2)+np.power(epos.y,2))*L/(m+np
     ↪ .cos(np.arcsin(np.sqrt(np.power(epos.x,2)+np.power(epos.y,2))/epos.R)))
     ↪ *np.sqrt(np.power(epos.x,2)+np.power(epos.y,2))/epos.R
2195 #recalculated detector coordinates, x
2196
2197 #x/(x^2+y^2)^(1/2)*L/(m+cos(arcsin((x^2+y^2)^(1/2)/R))*(x^2+y^2)^(1/2)/R
2198 epos.det_y2 = epos.y/np.sqrt(np.power(epos.x,2)+np.power(epos.y,2))*L/(m+np
     ↪ .cos(np.arcsin(np.sqrt(np.power(epos.x,2)+np.power(epos.y,2))/epos.R)))
     ↪ *np.sqrt(np.power(epos.x,2)+np.power(epos.y,2))/epos.R
2199 #recalculated detector coordinates, y
2200
2201 #y/(x^2+y^2)^(1/2)*L/(m+cos(arcsin((x^2+y^2)^(1/2)/R))*(x^2+y^2)^(1/2)/R
2202 #Geiser
2203 #Gault
2204
2205 #epos.det_x2 = epos.x*(1+L*10**6/(ICF*epos.R))/10**6
2206 #epos.det_y2 = epos.y*(1+L*10**6/(ICF*epos.R))/10**6
2207 #Bas et. al.
2208
2209 axes[0].hist2d(epos.det_x.get_values(), epos.det_y.get_values(), bins = 150)
2210 axes[1].hist2d(epos.det_x2.get_values(), epos.det_y2.get_values(), bins =
     ↪ 150)
2211
2212 #-----------------------------------------------------------------------
2213 #vergroe = epos.det_x.max()/epos.det_x2.max()
```

```
2214 vergroe =
     ↪  epos.ix[epos.det_x.argmax()].det_x/epos.ix[epos.det_x.argmax()].det_x2
2215 vergroe3 =
     ↪  epos.ix[epos.det_x.argmin()].det_x/epos.ix[epos.det_x.argmin()].det_x2
2216 #calculate magnification constants, for special positions of atoms
2217
2218 #-------------------------------------------------------------------------
     ↪  fits good
2219 vergroe4 =
     ↪  epos.ix[epos.det_y.argmin()].det_y/epos.ix[epos.det_y.argmin()].det_y2
2220 vergroe5 =
     ↪  epos.ix[epos.det_y.argmax()].det_y/epos.ix[epos.det_y.argmax()].det_y2
2221 #calculate magnification constants, for special positions of atoms
2222 #-------------------------------------------------------------------------
     ↪  fits good
2223 vergroe2 = (epos.det_x/epos.det_x2).mean()
2224 #calculate a mean magnification constant
2225 print(vergroe*L)
2226 print(vergroe2*L)
2227 print(vergroe3*L)
2228 print(vergroe4*L)
2229 print(vergroe5*L)
2230 #print calculated "virtual" flight lengths, used for reconstruction
     ↪  calibration
2231 #-------------------------------------------------------------------------
2232 #statisticx, xedge, yedge,binnummer = stats.binned_statistic_2d(
2233 #epos.det_x.get_values().flatten(),
2234 #epos.det_y.get_values().flatten(),
2235 #(epos.det_x-epos.det_x2).get_values().flatten(), statistic='mean', bins
     ↪  = [150,150])
2236 #calculate difference of det_x amd det_x2 and mean it space-resolved
     ↪  (binned statistics)
2237 #statisticx = np.nan_to_num(statisticx)
2238 #convert nan to zero
2239 #X,Y = np.meshgrid((xedge[:-1]+xedge[1:])/2,(yedge[:-1]+yedge[1:])/2)
2240 #create meshgrid for contour plot
2241 #axes[2].contourf(X.T,Y.T, statisticx,50)
2242 #creates contourplot
2243 #
2244 #statisticy, xedge, yedge,binnummer = stats.binned_statistic_2d(
2245 #epos.det_x.get_values().flatten(),
2246 #epos.det_y.get_values().flatten(),
2247 #(epos.det_y-epos.det_y2).get_values().flatten(), statistic='mean', bins
     ↪  = [150,150])
2248 #statisticy = np.nan_to_num(statisticy)
2249 #X,Y = np.meshgrid((xedge[:-1]+xedge[1:])/2,(yedge[:-1]+yedge[1:])/2)
```

```
2250 #axes[3].contourf(X.T,Y.T, statisticy,50)
2251 #calculate difference of det_y amd det_y2 and mean it space-resolved and
     ↪ contour plot it
2252 #-------------------------------------------------------------------------
2253 axes[0].set_title('det coord hit epos')
2254 axes[1].set_title('coor aus x y hit epos')
2255 axes[2].set_title(r'$\Delta x_{det}$ v. angepassten L')
2256 axes[3].set_title(r'$\Delta y_{det}$ v. angepassten L')
2257 [ax.set_xlabel(r'$\Delta x$') for ax in axes]
2258 [ax.set_ylabel(r'$\Delta y$') for ax in axes]
2259 #label axes of plots
2260
2261 #-------------------------------------------------------------------------
2262 #initial = np.ones((14))
2263 #popt, pcov = curve_fit(fitX, (epos.det_x, epos.det_y), epos.det_x2, p0 =
     ↪ initial)
2264 #popt, pcov = curve_fit(fitX, (epos.det_x[:1000], epos.det_y[:1000]),
     ↪ epos.det_x2[:1000], p0 = initial)
2265 #
2266 #popt, pcov = curve_fit(fitX3, (epos.det_x[:1000], epos.det_y[:1000]),
     ↪ epos.det_x2[:1000], p0 = initial)
2267 #
2268 #popt, pcov = curve_fit(FitXYlin, (epos.det_x[:1000], epos.det_y[:1000]),
     ↪ epos.det_x[:1000]-epos.det_x2[:1000], p0 = initial)
2269 #popt, pcov = curve_fit(FitXYlin, (epos.det_x, epos.det_y),
     ↪ epos.det_x-epos.det_x2, p0 = initial)
2270 #several fits to functions
2271 #
2272 ##test = fitX3((epos.det_x, epos.det_y),*popt)-epos.det_x2
2273 #test = FitXYlin((epos.det_x, epos.det_y),*popt)-(epos.det_x-epos.det_x2)
2274 #calculate residua
2275 #
2276 #statisticTest, xedge, yedge,binnummer = stats.binned_statistic_2d(
2277 #epos.det_x.get_values().flatten(),
2278 #epos.det_y.get_values().flatten(),
2279 #test.get_values().flatten(), statistic='mean', bins = [150,150])
2280 #statisticTest = np.nan_to_num(statisticTest)
2281 #X,Y = np.meshgrid((xedge[:-1]+xedge[1:])/2,(yedge[:-1]+yedge[1:])/2)
2282 #axes[0].contourf(X.T,Y.T, statisticTest,50)
2283 #mean and plot residua
2284 #
2285 #test = FitXYlin((epos.det_x, epos.det_y),*popt)
2286 #
2287 #statisticTest, xedge, yedge,binnummer = stats.binned_statistic_2d(
2288 #epos.det_x.get_values().flatten(),
2289 #epos.det_y.get_values().flatten(),
```

```
2290 #test.get_values().flatten(), statistic='mean', bins = [150,150])
2291 #statisticTest = np.nan_to_num(statisticTest)
2292 #X,Y = np.meshgrid((xedge[:-1]+xedge[1:])/2,(yedge[:-1]+yedge[1:])/2)
2293 #axes[1].contourf(X.T,Y.T, statisticTest,50)
2294 #mean and plot function values on binned values
2295 #
2296 #popt, pcov = curve_fit(FitXYlin2, (epos.det_x[:1000],
     ↪  epos.det_y[:1000]), epos.det_y[:1000]-epos.det_y2[:1000], p0 =
     ↪  initial)
2297 #popt, pcov = curve_fit(FitXYlin2, (epos.det_x, epos.det_y),
     ↪  epos.det_y-epos.det_y2, p0 = initial)
2298 #test = FitXYlin2((epos.det_x, epos.det_y),*popt)-(epos.det_y-epos.det_y2)
2299 #calculate difference of residua in x and y
2300 ##test = FitXYlin2((epos.det_x, epos.det_y),*popt)
2301 #
2302 #statisticTest, xedge, yedge,binnummer = stats.binned_statistic_2d(
2303 #epos.det_x.get_values().flatten(),
2304 #epos.det_y.get_values().flatten(),
2305 #test.get_values().flatten(), statistic='mean', bins = [150,150])
2306 #statisticTest = np.nan_to_num(statisticTest)
2307 #X,Y = np.meshgrid((xedge[:-1]+xedge[1:])/2,(yedge[:-1]+yedge[1:])/2)
2308 #axes[0].contourf(X.T,Y.T, statisticTest,50),
2309 #mean and plot difference of residua in x and y
2310 #-----------------------------------------------------------------------
2311 #poptX, pcovX = curve_fit(FitXYlin, (epos.det_x, epos.det_y),
     ↪  epos.det_x-epos.det_x2, p0 = initial)
2312 #poptY, pcovY = curve_fit(FitXYlin2, (epos.det_x, epos.det_y),
     ↪  epos.det_y-epos.det_y2, p0 = initial)
2313 #
2314 #poptX2, pcovX2 = curve_fit(FitXYlin, (epos.det_x, epos.det_y),
     ↪  epos.det_x2, p0 = initial)
2315 ##poptY, pcovY = curve_fit(FitXYlin2, (epos.det_x, epos.det_y),
     ↪  epos.det_y2, p0 = initial)
2316 ##
2317 #test = FitXYlin((epos.det_x, epos.det_y),*poptX2)-epos.det_x2
2318 #
2319 #statisticTest, xedge, yedge,binnummer = stats.binned_statistic_2d(
2320 #epos.det_x.get_values().flatten(),
2321 #epos.det_y.get_values().flatten(),
2322 #test.get_values().flatten(), statistic='mean', bins = [150,150])
2323 #statisticTest = np.nan_to_num(statisticTest)
2324 #X,Y = np.meshgrid((xedge[:-1]+xedge[1:])/2,(yedge[:-1]+yedge[1:])/2)
2325 #axes[0].contourf(X.T,Y.T, statisticTest,50)
2326 #mean and plot residua
2327 #-----------------------------------------------------------------------
2328 fig.tight_layout()
```

```
2329
2330 #ICF = 1.225
2331 #ICF = 1.4
2332 ICF = 1.225
2333 kf = 4.54
2334 kf0 = 4.54
2335
2336 ICF = 1.0178*ICF
2337 kf = 1.0178*kf
2338 #several used kf and ICF values, for new reconstructions based on
     ↪ recalculated
2339 #detector coordinates
2340
2341 FevAl = 19.
2342 eta = 0.37
2343 #detection efficiency
2344
2345 m = ICF-1.
2346 L = 382.
2347 #standard flight path IVAS
2348 epos = label_ions(epos,rrngs)
2349
2350 epos2 = epos.loc[:,['x','y','z','det_x2','det_y2','R','comp','dz','Da']]
2351 #create epos2 from epos values
2352
2353 for n,r in rrngs.iterrows():
2354     epos2.loc[(epos2.comp == r.comp),['dz']] = [r['vol']]
2355 #fill dz coloumn with atomic volumes dependent on species
2356 epos2.dz = epos2.dz.fillna(0.)
2357 #change nan entries to zero
2358
2359 SD = ((epos2.det_x2.max()-epos2.det_x2.min()))**2*np.pi/4.
2360 #calculated an approximate detector area
2361 SD = SD*0.807098
2362 #correct detector are so that reconstructions are identical,
2363 #value works for R21_09743-v01.epos
2364 epos2.dz =
     ↪ epos2.dz*L**2*kf**2*FevAl**2/(eta*ICF**2*(epos2.R*FevAl*kf0)**2*SD)
2365 #calculate z increments due to atomic volume
2366
2367 #epos2.x = epos2.det_x2/((1+L*10**6/(ICF*epos2.R))/10**6)
2368 #epos2.y = epos2.det_y2/((1+L*10**6/(ICF*epos2.R))/10**6)
2369 #Bas
2370
2371 theta_prime = np.arctan(np.sqrt(epos2.det_x2.pow(2)+epos2.det_y2.pow(2))/(L
     ↪ +ICF*epos2.R/10.**6))
```

```
2372 #calculate compressed angle
2373 epos2.x = epos2.det_x2/np.sqrt(epos2.det_x2.pow(2)+epos2.det_y2.pow(2))*epo
     ↪ s2.R*np.sin(theta_prime+np.arcsin(m*sin(theta_prime)))
2374 epos2.y = epos2.det_y2/np.sqrt(epos2.det_x2.pow(2)+epos2.det_y2.pow(2))*epo
     ↪ s2.R*np.sin(theta_prime+np.arcsin(m*sin(theta_prime)))
2375 #calculate x and y from recalculated detector coordinates
2376
2377 dzprime =
     ↪ epos2.R*(1-np.sqrt(1-(epos2.y.pow(2)+epos2.x.pow(2))/epos2.R.pow(2)))
2378 #calculate z changes due to radius
2379
2380 epos2.z = epos2.dz.cumsum()+dzprime
2381 #calculate z position
2382
2383 #epos2['Da'] = epos['Da']
2384 epos2 = label_ions(epos2,rrngs)
2385 #label ions
2386
2387 #epos.x = epos.x+epos2.x.max()-epos2.x.min()
2388 #epos2 = epos2.append(epos)
2389 #used to look at both reconstruction in 3D
2390
2391 #axes[2].clear()
2392 #axes[2].plot(np.linspace(0,np.pi),np.linspace(0,np.pi)+np.arcsin(m*np.si
     ↪ n(np.linspace(0,np.pi))),label='Gault -
     ↪ relation')
2393 #axes[2].set_xlabel('theta_prime')
2394 #axes[2].set_ylabel('theta')
2395 ##axes[2].plot(np.linspace(0,np.pi),np.linspace(0,np.pi)/ICF,label='theta
     ↪ _prime/ICF')
2396 #axes[2].plot(np.linspace(0,np.pi),np.linspace(0,np.pi)*ICF,label='theta_
     ↪ prime*ICF')
2397 #axes[2].legend()
2398 #plotting relations for comparison
2399
2400 volvis(epos2[epos2.comp =='Zn:1'])
2401 #3D view of Zn atoms of the new reconstruction
2402
2403 #-----------------------------------------------------------------------
2404 #axes[2].clear()
2405 #ICF0 = 1.225
2406 #kf0 = 4.54
2407 #
2408 #for i in np.linspace(0.5,2.5,10):
2409 #    ICF = i*ICF0
2410 #    kf = i*kf0
```

```
2411 #    theta_test =
     ↪ np.arctan(np.linspace(0,150,100)/(L+ICF*(6000./(kf*FevAl))/10.**6))
2412 #    dz_test =
     ↪ 6000./(kf*FevAl)*(1.-np.cos(theta_test+np.arcsin((ICF-1)*theta_test)))
2413 ##    print(kf/ICF)
2414 #    axes[2].plot(np.linspace(0,150,100),dz_test,label='ICF '+
     ↪ "{:.2f}".format(ICF) + ' kf '+ "{:.2f}".format(kf))
2415 #axes[2].legend(loc='best')
2416 #tests for the impact of changed absolute kf and ICF for a constant ratio
2417
2418 #axes[3].clear()
2419 #ICF = ICF0
2420 #for i in np.linspace(0.5,2.5,10):
2421 ##    ICF = i*ICF0
2422 #    kf = i*kf0
2423 #    theta_test =
     ↪ np.arctan(np.linspace(0,150,100)/(L+ICF*(6000./(kf*FevAl))/10.**6))
2424 #    dz_test =
     ↪ 6000./(kf*FevAl)*(1.-np.cos(theta_test+np.arcsin((ICF-1)*theta_test)))
2425 ##    print(kf/ICF)
2426 #    axes[3].plot(np.linspace(0,150,100),dz_test,label='kf '+
     ↪ "{:.2f}".format(kf))
2427 #axes[3].legend(loc='best')
2428 #tests for the impact of changed absolute kf and ICF for a constant ratio
```

8.5 ranging_kryo_proto.py

```
2429 from pyflann import *
2430 #from numpy import *
2431 import time
2432 from apt_importers import *
2433 #import scipy.linalg as lin
2434 from Vis import volvis
2435 from matplotlib import cm
2436 #from scipy import signal
2437 import ranging
2438 import itertools as it
2439 """
2440 used to plot mass-to-charge ratio (m/n) histogram and get possible
2441 combinations of elements for a specific m/n
2442 """
2443 pos = read_pos('/media/phillip/Volume1/Daten_PA/R21_09042/recons/korrICF/de
     ↪ fault/R21_09042-v02.pos')
2444
2445 deltaDalton = 0.02
2446 nBins = np.int((np.max(pos.Da)-np.min(pos.Da))/deltaDalton)
```

```
2447 fig = plt.figure()
2448 fig.set_size_inches(15.,10.)
2449 ax = fig.add_subplot(111)
2450 ax.set_title('mass-to-charge')
2451 H, edges = np.histogram(pos.Da, nBins)
2452 constCutoff = 0
2453 #H[H<constCutoff] = 0.
2454 xax = (edges[1:]+edges[:-1])/2.
2455 ax.plot(xax[H>constCutoff],H[H>constCutoff],'.-')
2456 #ax.plot(edges[:-1],H)
2457 #plots mass-to-charge ratio
2458 ax.set_xlabel('m/n [Da]')
2459 ax.set_ylabel('counts [1]')
2460
2461 #ax.set_xbound(17, 58)
2462 #ax.axvline(56.,label=r'£^{56}£Fe£^{+1}£',c='b',lw=1.5)
2463 #ax.axvline(56./2,label=r'£^{56}£Fe£^{+2}£',c='b',lw=1.5)
2464 #ax.axvline(56./3,label=r'£^{56}£Fe£^{+3}£',c='b',lw=1.5)
2465 #
2466 #ax.axvline(54.,label=r'£^{54}£Fe£^{+1}£',c='k',lw=1.5)
2467 #ax.axvline(54./2,label=r'£^{54}£Fe£^{+2}£',c='k',lw=1.5)
2468 #ax.axvline(54./3,label=r'£^{54}£Fe£^{+3}£',c='k',lw=1.5)
2469 #
2470 #ax.axvline(57.,label=r'£^{57}£Fe£^{+1}£',c='g',lw=1.5)
2471 #ax.axvline(57./2,label=r'£^{57}£Fe£^{+2}£',c='g',lw=1.5)
2472 #ax.axvline(57./3,label=r'£^{57}£Fe£^{+3}£',c='g',lw=1.5)
2473 #ax.legend(loc='best')
2474
2475 ax.set_yscale("log")
2476
2477 pathRange = '/media/phillip/Volume1/13_rangeFiles/R34_05935_AlOH_V.rrng'
2478
2479 ions,rrngs = read_rrng(pathRange)
2480 rrngs = rrngs.sort('lower')
2481 #labels ranged ions
2482
2483 for i, wert in enumerate(rrngs.lower):
2484     ax.axvline(wert, color='black')
2485     ax.axvline(rrngs.upper[i], color='r')
2486     ax.annotate(rrngs.comp[i].replace('Name:',''), xy=(wert, 100. +
      ↪  (i%4)*10**(i%4+1)))
2487 #marks labeled ranges
2488
2489 pos = label_ions(pos, rrngs)
2490 #lpos=lpos[lpos.comp != '']
2491 #name, comp = getComp(lpos)
```

```
#calculates composition
#------------------------------------------------------------------------

#Atoms = [1., 16., 24., 27., 28., 51., 52., 63., 69.]
#Atoms = [1., 16., 24., 27., 28., 63.]
#Atoms = [1., 16., 24., 27., 28., 51., 63.]
#
#eps = 0.01
##molarTarget = 33
#molarTarget = 17
#
#for j in range(1,5):
#    iteratorAtoms = it.combinations_with_replacement(Atoms,j)
#
#    for i in iteratorAtoms:
#        summe = np.sum(i)
#        for z in range(1,4):
#            if summe/z < molarTarget+eps and summe/z > molarTarget-eps:
#                print(i, z)
#used to calculate possible combinations of elements (Atoms) to fit to a
↪  specific
#mass-to-charge ratio, molarTarged
```

8.6 multiple_ion_analysis.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 21 11:40:09 2016

@author: phillip
"""
from pyflann import *
#from numpy import *
import time
from apt_importers import *
#import scipy.linalg as lin
#from Vis import volvis
from matplotlib import cm
from scipy.special import gamma
from scipy.misc import factorial
import itertools as it
#import ranging
#from Vis import volvis
def getCorrComp(path,pathRange):
    for i in range(len(path)):
```

```
2533            pfad = '/'.join(path[i].split('/')[:-4])+ '/auswertung/'+
                ↪  path[i].split('/')[-1][:-4] + "_comp_corr.txt"
2534            ions, rrngs = read_rrng(pathRange[i])
2535            pos = read_pos(path[i])
2536            fig = getCompPos(pos, rrngs, pfad)
2537    #         fig.savefig('/'.join(path[i].split('/')[:-4])+ '/auswertung/'+
        ↪  path[i].split('/')[-1][:-4] + "_bg_fit.pdf")
2538        return
2539    def getAlRelList(path, pathRange):
2540        AlRel = np.zeros((len(path),2))
2541        for i in range(len(path)):
2542            ions, rrngs = read_rrng(pathRange[i])
2543            pos = read_pos(path[i])
2544            lpos = label_ions(pos, rrngs)
2545            AlRel[i] = getAlRel(lpos)
2546
2547        return AlRel
2548    def getAlRel(lpos):
2549        nrAlRanges = rrngs[rrngs.comp == 'Al:1'].comp.count()
2550        if nrAlRanges == 3:
2551            Al1 = lpos.ix[(lpos.Da > rrngs[rrngs.comp == 'Al:1'].ix[2].lower) &
                ↪  (lpos.Da < rrngs[rrngs.comp ==
                ↪  'Al:1'].ix[2].upper)].comp.count()
2552            Al2 = lpos.ix[(lpos.Da > rrngs[rrngs.comp == 'Al:1'].ix[1].lower) &
                ↪  (lpos.Da < rrngs[rrngs.comp ==
                ↪  'Al:1'].ix[1].upper)].comp.count()
2553            Al3 = lpos.ix[(lpos.Da > rrngs[rrngs.comp == 'Al:1'].ix[0].lower) &
                ↪  (lpos.Da < rrngs[rrngs.comp ==
                ↪  'Al:1'].ix[0].upper)].comp.count()
2554
2555        elif nrAlRanges == 2:
2556            Al1 = lpos.ix[(lpos.Da > rrngs[rrngs.comp == 'Al:1'].ix[1].lower) &
                ↪  (lpos.Da < rrngs[rrngs.comp ==
                ↪  'Al:1'].ix[1].upper)].comp.count()
2557            Al2 = lpos.ix[(lpos.Da > rrngs[rrngs.comp == 'Al:1'].ix[0].lower) &
                ↪  (lpos.Da < rrngs[rrngs.comp ==
                ↪  'Al:1'].ix[0].upper)].comp.count()
2558            Al3 = 0
2559        else:
2560            Al2=0.
2561            Al1=1.
2562            Al3=0.
2563
2564        summe = Al1 + Al2 + Al3
2565        print(float(Al2)/summe)
2566        print(float(Al3)/summe)
```

```
2567     print("\n")
2568     #DEBUGGING Info
2569     return float(Al2)/summe, float(Al3)/summe
2570 def countIons(d):
2571     lower = 58.8
2572     upper = 59.2
2573     d.loc[:,'in'] = d.loc[(d.Da < upper) & (d.Da > lower)].Da.count() > 0
2574     return d
2575 """
2576 saves compositions of path list and pathRange list to analysis folders,
2577 compares kNN of IVAS to script data,
2578 creates a saxey plot and lot mass-to-charge state ratios for multiples,
2579 prints possible combinations of ions for unknown peak,
2580 prints positions of multiple hits for specific element onto detector
    ↪ hitmap
2581 """
2582
2583 path = []
2584 path.append('/media/phillip/Volume1/DatenIVAS/R34_04231/recons/recon-v01/de
    ↪ fault/R34_04231-v02.pos')
2585 pathRange = []
2586 pathRange.append('/media/phillip/Volume1/13_rangeFiles/R34_04231root.RRNG')
2587
2588
2589 getCorrComp(path,pathRange)
2590 #writes compositions to analysis directories
2591
2592 #AlRel = getAlRelList(path, pathRange)
2593
2594 fig = plt.figure()
2595 conv = 2.54
2596 #1 inch = 2.54cm
2597 breite=15
2598 hoehe=25
2599 fig.set_size_inches(breite,hoehe)
2600 ax = fig.add_subplot(211)
2601 ax2 = fig.add_subplot(212)
2602 #creates figure and adds subplots
2603
2604 #ax = fig.add_subplot(211)
2605 #ax2 = fig.add_subplot(212)
2606
2607 #ax.plot(AlRel[:,0])
2608 #ax2.plot(AlRel[:,1])
2609 #-----------------------------------------------------------------------
```

```
2610 IVAS = '/media/phillip/Volume1/DatenIVAS/R34_04231/auswertung/R34_04231-v02
     ↪  _NN10IVAS.csv'
2611 #IVAS exported data
2612
2613 pythonNN = '/media/phillip/Volume1/DatenIVAS/R34_04231/auswertung/R34_04231
     ↪  -v02Sitarget_Si_kNN_Data.txt'
2614 #NN data via script
2615
2616 A = np.loadtxt(IVAS, delimiter=',', skiprows=8)
2617 stutz = A[:,0]+A[:,1]/100.
2618 vals = A[:,2]
2619 rand = A[:,3]
2620
2621 [stutz2, a, a3, comp0std] = np.loadtxt(pythonNN, delimiter=',')
2622
2623 ax.plot(stutz, vals, label = 'vals IVAS')
2624 ax.plot(stutz, rand, label = 'rand IVAS')
2625
2626 ax.plot(stutz2, a3, label = 'vals python')
2627 ax.plot(stutz2, a, label = 'rand python')
2628
2629 ax.legend(loc = 'best')
2630 #plots comparison of IVAS and script k nearest neighbor distances
2631
2632 print(np.sum(a3)-np.sum(vals), 'difference in counts for values')
2633 print(np.sum(a)-np.sum(rand), 'difference in counts for random values')
2634 #DEBUGGING Info
2635 #---------------------------------------------------------------------
2636 ax.clear()
2637 epos = read_epos('/media/phillip/Volume1/PD_august_2018_GallenN/R34_05935/r
     ↪  econs/recon-v02/default/R34_05935-v02.epos')
2638 pathRange = '/media/phillip/Volume1/13_rangeFiles/R34_05935_AlOH_V.rrng'
2639
2640 multiples = epos[epos.ipp !=1]
2641 #selects multiple hits
2642
2643 #multiples = epos[epos.ipp ==2]
2644 #doubles
2645 #multiples = epos[epos.ipp ==1]
2646 #singles
2647
2648 ions,rrngs = read_rrng(pathRange)
2649
2650 deltaDalton = 0.02
2651 nBins = np.int32((multiples.Da.max() - multiples.Da.min()) / deltaDalton)
2652 H, edges = np.histogram(multiples.Da, bins = nBins)
```

```
2653 rrngs = rrngs.sort('lower')
2654 ax.plot((edges[1:]+edges[:-1])/2., H)
2655 ax.set_yscale('log')
2656
2657 for i, wert in enumerate(rrngs.lower):
2658     ax.axvline(wert, color='black')
2659     ax.axvline(rrngs.upper[i], color='r')
2660     ax.annotate(rrngs.comp[i].replace('Name:',''), xy=(wert, 100. +
     ↪  (i%4)*10**(i%4+1)))
2661 #creates mass-to-charge ratio histogram and plots it
2662 #-----------------------------------------------------------------------
2663 #ax.clear()
2664 upperDa = 100.
2665 numberBins = np.ceil(upperDa/deltaDalton)
2666 edges = np.linspace(0, upperDa, numberBins)
2667
2668 #c, d = np.histogramdd(array([[np.nan ,np.nan]]), bins = (edges, edges))
2669 entryLang = 0
2670 for i in range(1,multiples.ipp.max()+1):
2671     n = multiples.ipp[multiples.ipp == i].count()
2672     entryLang += n * i*(i-1)/2.
2673 #number of combinations of 2 atoms in all multiples
2674 pairs = np.zeros((entryLang, 2))
2675 #creates empty array
2676
2677 counter = 0
2678 counterOld = counter
2679 counterPair = 0
2680
2681 while counter < multiples.ipp.count():
2682     counterOld = counter
2683     n = multiples.ipp.iloc[counter]
2684     counter = counter + n
2685 #     c += np.histogramdd(np.asarray(list(it.combinations(multiples.Da.ilo
     ↪  c[counterOld:counter],2))), bins = (edges,
     ↪  edges))[0]
2686     pairs[counterPair:counterPair+n*(n-1)/2,:] =  np.asarray(list(it.combin
     ↪  ations(multiples.Da.iloc[counterOld:counter],2)))
2687     counterPair = counterPair + n*(n-1)/2
2688 #calculate pair combinations for all multiples
2689
2690 #H, E = np.histogramdd(pairs, bins = (edges, edges))
2691 #X, Y = np.meshgrid((E[0][:-1]+E[0][1:])/2.,(E[1][:-1]+E[1][1:])/2.)
2692 #ax.contourf(X, Y, H, 100)
2693
2694 ax2.scatter(pairs[:,0],pairs[:,1], alpha=0.3, s=5, lw=0, cmap=cm.gray)
```

```
2695 ax2.scatter(pairs[:,1],pairs[:,0], alpha=0.3, s=5, lw=0, cmap=cm.gray)
2696 #ax2.set_xbound(26,30)
2697 ax2.set_xbound(0, 75)
2698 ax2.set_ybound(0.,75)
2699 ax2.grid(True)
2700
2701 ax2.set_xlabel('m$_1$ [Da]', fontsize = 22)
2702 ax2.set_ylabel('m$_2$ [Da]', fontsize = 22)
2703 ax2.set_title('ion correlation diagram', fontsize = 22)
2704 #plots and labels saxey plot
2705
2706 #HDa, edgesDa = np.histogram(multiples.Da,
     ↪  bins=np.ceil(multiples.Da.max()/deltaDalton))
2707 #ax2.plot((edgesDa[:-1]+edgesDa[1:])/2., HDa)
2708 #ax2.set_yscale('log')
2709 #ax.set_xbound(26,30)
2710 ax.set_xbound(0.,75)
2711
2712 #numberlines = 50
2713 #X,Y = np.meshgrid((edges[:-1]+edges[1:])/2., (edges[:-1]+edges[1:])/2.)
2714 #Z = np.histogramdd(pairs, bins = (edges, edges))[0]
2715 #cnt = ax2.contourf(X,Y,Z,numberlines, cmap=cm.gray_r)
2716
2717 #---------------------------------------------------------------------
2718 ax.clear()
2719 multiples.loc[:,'id'] = multiples.ipp.cumsum()
2720 mg = multiples.groupby('id')
2721 multiples = mg.apply(hitlabel)
2722
2723 #for i in range(2,multiples.ipp.max()+1):
2724 for i in range(2,6):
2725     auswahl = multiples.ix[multiples.hn == i]
2726     H, edges = np.histogram(auswahl.Da, bins = nBins)
2727     ax.plot((edges[1:]+edges[:-1])/2., H, label = str(i))
2728 ax.set_yscale('log')
2729 ax.legend(loc='best')
2730
2731
2732 auswahl = multiples.ix[multiples.hn == 3]
2733 multiNeu = auswahl.groupby('id').apply(countIons)
2734 HNeu, edgesNeu = np.histogram(multiNeu.ix[multiNeu['in']].Da, bins = nBins)
2735 ax.plot((edgesNeu[1:]+edgesNeu[:-1])/2., HNeu)
2736 ax.set_xbound(0.,75)
2737
2738 ax.set_xlabel('mass-to-charge [Da]', fontsize = 22)
2739 ax.set_title('multiple ion events', fontsize = 22)
```

```
ax.set_ylabel('counts', fontsize = 22)
#plots a specified multple mass-to-charge state
##-----------------------------------------------------------------------
##Atoms = [9., 13.5, 27., 16., 18., 21.5, 23.33, 34.5, 35, 35.33, 43.,
↪  44., 59., 52., 69.,]
#Atoms = [1., 16., 18., 24., 27., 28., 63.,  69.]
#eps = 0.01
#molarTarget = 43.
#
#for j in range(1,5):
#    iteratorAtoms = it.combinations_with_replacement(Atoms,j)
#    for i in iteratorAtoms:
#        summe = np.sum(i)
#        for z in range(1,4):
#            if summe/z < molarTarget+eps and summe/z > molarTarget-eps:
#                print(i, z)
##prints possible combinations for molarTarges
##-----------------------------------------------------------------------
#multiples = label_ions(multiples, rrngs)
#multipleSi = multiples.loc[(multiples.comp == 'Si:1') & (multiples.ipp
↪  == 2)]
#ax2.clear()
##ax.clear()
#multipleGa = multiples.loc[multiples.comp == 'Ga:1']
#ax2.hist2d(epos.det_x.get_values(), epos.det_y.get_values(), bins = 100)
#ax2.scatter(multipleSi.det_x.get_values(),
↪  multipleSi.det_y.get_values(),lw=0.,c = 'black')
#ax2.scatter(multipleGa.det_x.get_values(),
↪  multipleGa.det_y.get_values(),lw=0.,c = 'gray')
##plots multiple Si hits and multiple Ga hits to detector hitmap
```

8.7 proto_function_RDF_data.py

```
from pyflann import *
#from numpy import *
import time
from apt_importers import *
#import scipy.linalg as lin
#from Vis import volvis
from matplotlib import cm
#from scipy import signal
import ranging
from scipy.stats import ks_2samp
from scipy import stats
"""
```

```
2778 function to print the RDF and cumulative RDF to txt and create pdf
     ↪ figures,
2779 pos files are chosen via adding to path list and rrrng via adding to
     ↪ pathRange
2780 """
2781 def writeRDFkNN_data(lpos, path, kNN = 10):
2782     set_distance_type('euclidean')
2783     flannObj = FLANN()
2784     #creates FLANN object
2785 
2786     dataset = []
2787     element = []
2788     dataset.append(lpos.ix[lpos.comp == 'Mg:1',1:4].get_values())
2789     element.append('Mg')
2790     dataset.append(lpos.ix[lpos.comp == 'Si:1',1:4].get_values())
2791     element.append('Si')
2792 #    select atoms for analysis
2793     #-------------------------------------------------------------------
2794     fig2 = plt.figure()
2795     conv = 2.54
2796     #1 inch = 2.54cm
2797     breite=15/conv
2798     hoehe=24/conv
2799     fig2.set_size_inches(breite,hoehe)
2800     #-------------------------------------------------------------------
2801     #creates figure to plot
2802 
2803     #kNN0 = 5S
2804 #    kNN = 10
2805 #    kNN = 5
2806 #    binSize = 200
2807     binSize = 100
2808     nBinRDF = 100
2809 #    nBinRDF = 250
2810 
2811     #binning
2812 
2813     #ranges = (0.,10)
2814     ranges = (0.,7)
2815     #ranges of binning for RDF
2816 
2817     ax2 = fig2.add_subplot(411)
2818     ax3 = fig2.add_subplot(412)
2819     ax4 = fig2.add_subplot(413)
2820     ax5 = fig2.add_subplot(414)
2821     #adds subplots
```

```
2822      compwiederh =40
2823      #number of drawings for the random comparator
2824      rRDF = 2.
2825      rRDF = 5.
2826      #maximum for RDF radius
2827
2828      deltaBin = rRDF / nBinRDF
2829      #stutz = (b[1:]+b[:-1])/2.
2830
2831      colors = np.linspace(0,1,len(dataset)**2)
2832      lang= len(dataset)
2833      map0 = cm.nipy_spectral
2834
2835      threshcounts = 0
2836
2837      a0 = np.zeros(nBinRDF)
2838      comp0 = np.zeros(binSize)
2839
2840      a0List = np.zeros((compwiederh, nBinRDF))
2841      comp0List = np.zeros((compwiederh, binSize))
2842      #initialize lists to hold RDF values for random distributions
2843      output = ""
2844      #empty string
2845
2846      for j, target in enumerate(dataset):
2847          for i,wert in enumerate(dataset):
2848  #                 if j != i:
2849  #                     continue
2850      #         try:
2851                    #calculate for every Me_i x Me_j ... e.g. Mg, Si: Mg-Mg,
                      ↪  Mg-Si, Si-Mg, Si-Si
2852                    ax2.clear()
2853                    ax3.clear()
2854                    ax4.clear()
2855                    ax5.clear()
2856                    #clears the subplots
2857                    if i == j:
2858                        linest = '--'
2859                    else:
2860                        linest = '-'
2861                    #selects linestyle for cross and auto correlation
2862                    params = flannObj.build_index(target, algorithm=4,
                        ↪  target_precision=1., log_level = "info") #algorithm=4
                        ↪  selects kdtree single
2863                    #builds kd tree, where neares points are searched (target)
2864
```

```
2865                    a0 = np.zeros(nBinRDF)
2866                    #RDF values
2867                    comp0 = np.zeros(binSize)
2868                    #k NN values
2869
2870                    comp0List = np.zeros((compwiederh, binSize))
2871                    #k NN values list, for random comparator
2872                    a0List = np.zeros((compwiederh, nBinRDF))
2873                    #list for RDF values, for random comparator
2874                    cumsumList = np.zeros((compwiederh, nBinRDF))
2875                    #list for cumulative RDF, for random comparator
2876
2877                    b1, a1 = calcRDF(flannObj, params, wert, rRDF, nBinRDF,
                        ↪  target, stepSize=10)
2878                    #calculate existing RDF
2879                    a3, b = createNNHist(kNN, wert, flannObj, params, ranges,
                        ↪  binSize)
2880                    #calculate existing kNN distribution
2881                    cumRDF_meas = np.cumsum(a1)
2882                    #cumulative sum of existing data
2883                    counterWiederh = 0
2884                    #counter for repetitions of random drawings
2885
2886            #         ax2.plot(b, a/np.float(np.sum(a)),linest,
            ↪  label=element[j]+ " - "+ element[i],c = map0(colors[j*lang+
            ↪  i]), lw = 2.)
2887                    for k in range(0,compwiederh):
2888                        if i!= j:
2889  #                          comparator = lpos.ix[lpos.comp !=
    ↪  element[j]+':1'].sample(np.size(wert,0)).ix[:,1:4].get_values()
2890                            #OLD
2891                            comparator = lpos.ix[lpos.comp !=
                                ↪  element[j]+':1'].sample(np.size(wert,0)).loc[:,
                                ↪  ['x','y','z']].get_values()
2892                        else:
2893  #                          comparator =
    ↪  lpos.sample(np.size(wert,0)).ix[:,1:4].get_values()
2894                            #OLD
2895                            comparator = lpos.sample(np.size(wert,0)).loc[:,['x
                                ↪  ','y','z']].get_values()
2896                            flannObj.delete_index()
2897                            #delete old target
2898                            params = flannObj.build_index(comparator,
                                ↪  algorithm=4, target_precision=1., log_level =
                                ↪  "info") #algorithm=4 selects kdtree single
```

```
2899                    #everytime i==j is True a new target is built
                        ↪
2900
2901                a0List[counterWiederh,:] = calcRDF(flannObj, params,
                    ↪  comparator , rRDF, nBinRDF, target, stepSize=10)[1]
2902                #adds RDF into list
2903
2904                comp0List[counterWiederh,:] = createNNHist(kNN,
                    ↪  comparator, flannObj, params, ranges, binSize)[0]
2905                #adds kNN into list
2906                cumsumList[counterWiederh,:] =
                    ↪  np.cumsum(a0List[counterWiederh,:])
2907                #calculate cumulative RDF of current one and add into
                    ↪  list
2908
2909                counterWiederh += 1
2910                #increase counter
2911
2912            a0 = np.mean(a0List, axis = 0, dtype = float64)
2913            a0std= np.std(a0List, axis = 0, dtype = float64)
2914            #mean and calculate standard deviation for random
                ↪  comparator, RDF
2915
2916            cumRDF = np.mean(cumsumList, axis = 0, dtype = float64)
2917            cumRDF_std = np.std(cumsumList, axis = 0, dtype = float64)
2918            #mean and calculate standard deviation for random
                ↪  comparator, cumulative RDF
2919
2920            comp0 = np.mean(comp0List, axis = 0, dtype = float64)
2921            comp0std = np.std(comp0List, axis = 0, dtype = float64)
2922            #mean and calculate standard deviation for random
                ↪  comparator, kNN
2923
2924            a = comp0
2925            #adds alias for kNN distribution
2926
2927            ax2.errorbar(b1, a1/a0, a1/np.power(a0,2)*a0std,
                ↪  label=element[j]+ " rel. RDF - "+ element[i],c =
                ↪  map0(colors[j*lang+ i]), lw = 1.5)
2928            #plots existing RDF divided by random comparator to
                ↪  subplot
2929            ax2.axhline(1., linestyle = '-', c = map0(colors[j*lang+
                ↪  i]), lw=1.5)
2930
2931            delta = a1-a0
```

```
2932                delta[np.logical_and(delta<=threshcounts, -delta <=
                    ↪  threshcounts)] = 0.
2933                ax4.plot(b1, delta,'-x', label=element[j]+ " counts - comp
                    ↪  "+ element[i],c = map0(colors[j*lang+ i]), lw = 1.5)
2934                #plots difference of existing RDF to random comparator
2935
2936                ax5.plot(b1, a1, c = map0(colors[j*lang+ i]),label =
                    ↪  'data', lw = 1.5)
2937                ax5.plot(b1, a0, c = map0(colors[j*lang+ i]), label =
                    ↪  'comp', lw = 0.5)
2938                #plots existing and random to subplot
2939
2940                ax5.set_title('RDF')
2941                testVar = np.sqrt(np.sum(np.power(a1-a0,2)))/np.sum(a0)
2942                output += str(testVar) + "\n"
2943                print(testVar)
2944                #prints some testing information
2945
2946                stutz = (b[1:]+b[:-1])/2.
2947                ax3.plot(stutz, a3/np.float(np.sum(a3)),linest,
                    ↪  label=element[j]+ " - " + element[i] +" kNN "
                    ↪  +str(kNN),c = map0(colors[j*lang+ i]), lw = 1.5)
2948                ax3.axvline(stutz[getMean(a3)], linestyle = linest, c =
                    ↪  map0(colors[j*lang+ i]), lw=1.5)
2949                #plots existing kNN distribution, normed
2950
2951                print(stutz[getMean(a3)], element[j], 'kNN', element[i])
2952
2953                output += str([stutz[getMean(a3)], element[j], 'kNN',
                    ↪  element[i]]) + "\n"
2954                #output is currently not used anymore
2955
2956                ax3.errorbar(stutz, a/np.float(np.sum(a)), yerr =
                    ↪  comp0std/np.sum(a), label=element[j]+ " - " +
                    ↪  element[i] +" kNN comp " +str(kNN),c =
                    ↪  map0(colors[j*lang+ i]), lw = 0.5)
2957                ax3.axvline(stutz[getMean(a)], linestyle = '-.', c =
                    ↪  map0(colors[j*lang+ i]), lw=0.5)
2958                print(stutz[getMean(a)], element[j], 'kNN', element[i],
                    ↪  'comp')
2959
2960                output += str([stutz[getMean(a)], element[j], 'kNN',
                    ↪  element[i], 'comp']) + "\n"
2961        #           print(j)
2962                #DEBUGGING Info
2963
```

```
2964                    ax2.legend(loc='best',fontsize = 8)
2965                    ax3.legend(loc='best',fontsize = 8)
2966                    ax4.legend(loc='best',fontsize = 8)
2967                    ax5.legend(loc='best',fontsize = 8)
2968
2969                    ax2.set_title('RDF')
2970                    ax3.set_title('kNN-distribution')
2971                    ax4.set_title(r'(RDF-comp)')
2972                    #labeling
2973
2974                    fig2.savefig('/'.join(path.split('/')[:-4])+
                        ↪  '/auswertung/'+ path.split('/')[-1][:-4] +element[j] +
                        ↪  'target_' + element[i] + '_notNorm_'+str(kNN)+'.eps')
2975                    np.savetxt('/'.join(path.split('/')[:-4])+ '/auswertung/'+
                        ↪  path.split('/')[-1][:-4] +element[j] + 'target_' +
                        ↪  element[i] + '_RDF_Data.txt', np.array([b1, a0, a1,
                        ↪  a0std]).T, delimiter = ',')
2976                    np.savetxt('/'.join(path.split('/')[:-4])+ '/auswertung/'+
                        ↪  path.split('/')[-1][:-4] +element[j] + 'target_' +
                        ↪  element[i] + '_kNN_Data_'+str(kNN)+'.txt',
                        ↪  np.array([stutz, a, a3, comp0std]).T, delimiter = ',')
2977                    np.savetxt('/'.join(path.split('/')[:-4])+ '/auswertung/'+
                        ↪  path.split('/')[-1][:-4] +element[j] + 'target_' +
                        ↪  element[i] + '_cumRDF.txt', np.array([b1, cumRDF,
                        ↪  cumRDF_meas, cumRDF_std]).T, delimiter = ',')
2978                    #normally used
2979
2980 #                   fig2.savefig('/'.join(path.split('/')[:-4])+
     ↪  '/auswertung/'+ path.split('/')[-1][:-4] +element[j] + 'targetTest2_'
     ↪  + element[i] + '_notNorm_'+str(kNN)+'.eps')
2981 #                   np.savetxt('/'.join(path.split('/')[:-4])+
     ↪  '/auswertung/'+ path.split('/')[-1][:-4] +element[j] + 'targetTest2_'
     ↪  + element[i] + '_RDF_Data.txt', np.array([b1, a0, a1, a0std]).T,
     ↪  delimiter = ',')
2982 #                   np.savetxt('/'.join(path.split('/')[:-4])+
     ↪  '/auswertung/'+ path.split('/')[-1][:-4] +element[j] + 'targetTest2_'
     ↪  + element[i] + '_kNN_Data_'+str(kNN)+'.txt', np.array([stutz, a, a3,
     ↪  comp0std]).T, delimiter = ',')
2983 #                   np.savetxt('/'.join(path.split('/')[:-4])+
     ↪  '/auswertung/'+ path.split('/')[-1][:-4] +element[j] + 'targetTest2_'
     ↪  + element[i] + '_cumRDF.txt', np.array([b1, cumRDF, cumRDF_meas,
     ↪  cumRDF_std]).T, delimiter = ',')
2984                    #for testing purposes
2985
2986                    flannObj.delete_index()
2987                    #delete FLANN object
```

```
2988
2989 #    f = open('/'.join(path.split('/')[:-4])+ '/auswertung/'+
     ↪ path.split('/')[-1][:-4]+
     ↪ 'output_KS'+time.strftime("%Y_%m_%d_%H_%M_%S")+'.txt', 'w')
2990 #    f.write(output)
2991 #    f.close()
2992     #OLD
2993
2994     saveComp(lpos, '/'.join(path.split('/')[:-4])+ '/auswertung/'+
     ↪  path.split('/')[-1][:-4] + "_comp.txt")
2995     #save composition to analysis folder of pos file
2996     return
2997
2998
2999 path = []
3000 path.append('/media/phillip/Volume1/Alice_Messungen/R21_10253/recons/recon-
     ↪ v02/default/R21_10253-v02_cut.pos')
3001 #-----------------------------------------------------------------------
3002 #analyzed pos paths
3003
3004 pathRange = []
3005 pathRange.append('/media/phillip/Volume1/13_rangeFiles/R34_05935_AlOH_V.rrn
     ↪ g')
3006 #-----------------------------------------------------------------------
3007 #used range files
3008
3009 for i in range(len(path)):
3010     pos = read_pos(path[i])
3011     ions,rrngs = read_rrng(pathRange[i])
3012
3013     lpos = label_ions(pos, rrngs)
3014     lpos = lpos[lpos.comp != '']
3015     writeRDFkNN_data(lpos, path[i], kNN = 5)
3016 #calculate RDF and plots figure for every pos file in list
```

8.8 Vis.py

```
3017 from apt_importers import *
3018 import numpy as np
3019 import matplotlib.pyplot as plt
3020
3021 import pandas as pd
3022 import math
3023 import ranging
3024 from matplotlib import cm
3025
```

```
3026 def volvis(pos, size=2, alpha=1):
3027     """Displays a 3D point cloud in an OpenGL viewer window.
3028     If points are not labelled with colours, point brightness
3029     is determined by Da values (higher = whiter)"""
3030     from vispy import app,scene, mpl_plot
3031     import numpy as np
3032     import sys
3033     import matplotlib
3034     import re
3035 
3036     canvas = scene.SceneCanvas('APT Volume',keys='interactive',
         ↪  bgcolor='#ffffff')
3037 #    canvas = scene.SceneCanvas('APT Volume',keys='interactive')
3038     view = canvas.central_widget.add_view()
3039     view.camera = scene.TurntableCamera(up='z')
3040 
3041     cpos = pos.loc[:,['x','y','z']].values
3042 
3043     if 'colour' in pos.columns:
3044         colours =
             ↪  np.asarray(list(pos.colour.apply(matplotlib.colors.hex2color)))
3045     else:
3046        Dapc = pos.Da / np.max(pos.Da)
3047        colours = np.array(zip(Dapc,Dapc,Dapc))
3048 
3049 #    colval = np.linspace(0,1.,len(pos.groupby('comp')))
3050 #    counter = 0
3051 #    for name, group in pos.groupby('comp'):
3052 #        group.color = matplotlib.colors.rgb2hex(cm.jet(colval[counter]))
3053 #    colours =
     ↪  np.asarray(list(pos.colour.apply(matplotlib.colors.hex2color)))
3054 #    Dapc = np.zeros(np.size(lpos,0))
3055 #    colours = cm.jet(Dapc)[:,:-1]
3056 
3057 
3058     if alpha is not 1:
3059         np.hstack([colours, np.array([0.5] * len(colours))[...,None]])
3060 
3061     p1 = scene.visuals.Markers()
3062     p1.set_data(cpos, face_color=colours, edge_width=0, size=size)
3063 
3064     view.add(p1)
3065 
3066     # make legend
3067     ions = []
3068     cs = []
```

```
3069        for g,d in pos.groupby('comp'):
3070            ions.append(re.sub(r':1?|\s?','',d.comp.iloc[0]))
3071            cs.append(matplotlib.colors.hex2color(d.colour.iloc[0]))
3072        ions = np.array(ions)
3073        cs = np.asarray(cs)
3074
3075        pts = np.array([[20] * len(ions), np.linspace(20,20*len(ions),
            ↪ len(ions))]).T
3076        tpts = np.array([[30] * len(ions), np.linspace(20,20*len(ions),
            ↪ len(ions))]).T
3077
3078        legb = scene.widgets.ViewBox(parent=view, border_color='red',
            ↪ bgcolor='k')
3079    #    legb = scene.widgets.ViewBox(parent=view, border_color='red',
        ↪ bgcolor='#ffffff')
3080        legb.pos = 0,0
3081        legb.size = 100,20*len(ions)+20
3082
3083        leg = scene.visuals.Markers()
3084        leg.set_data(pts, face_color=cs)
3085        legb.add(leg)
3086
3087        legt = scene.visuals.Text(text=ions,pos=tpts,color='#ffffff',
            ↪ anchor_x='left', anchor_y='center', font_size=10)
3088
3089        legb.add(legt)
3090
3091        # show viewer
3092        canvas.show()
3093
3094        if sys.flags.interactive == 0:
3095            app.run()
```

8.9 largeSDM.py

```
3096 from pyflann import *
3097 import time
3098 from apt_importers import *
3099 from Vis import volvis
3100 from mpl_toolkits.mplot3d import Axes3D
3101
3102 numberPlots = 6
3103 fig, axes = plt.subplots(numberPlots/2,2, figsize = (10,15))
3104
3105 pos = read_pos('/media/phillip/Volume1/Daten_PA/R21_08675/auswertung/older_
    ↪ Evaluation/R21_08675-v02.pos')
```

```
3106 pathRange = '/media/phillip/Volume1/13_rangeFiles/R21_08675root.RRNG'
3107 ions,rrngs = read_rrng(pathRange)
3108 pos = label_ions(pos, rrngs)
3109 pos2 = read_pos('/media/phillip/Volume1/Daten_PA/R21_08675/auswertung/older
     ↪  _Evaluation/pole311_08675_v02_OLD.pos')
3110 pos2 = label_ions(pos2, rrngs)
3111 axes = axes.flatten()
3112
3113 cluster = pd.read_pickle('/media/phillip/Volume1/Daten_PA/R21_08675/auswert
     ↪  ung/older_Evaluation/R21_08675-v02_cluster2.pkl')
3114
3115 #deltasAl = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),20)
3116 deltasAl = getDeltasSDMLarge(pos2.loc[pos2.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),20)
3117
3118 deltasMg = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),20)
3119 deltasSi = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Si:1',['x','y','z']].get_values(),20)
3120
3121 deltaZMax = 2.
3122 #-----------------------------------------------------DIRECTIONS
3123 psi = -0.076666666666663941
3124 theta = 0.19666666666667121
3125 #002 Pole dhkl = 0.186 gemessen an Kugel ROI
3126 #-----------------------------------------------------DIRECTIONS END
3127 maxAnglePsi = np.pi/2.
3128 maxAngleTheta = np.pi/2.
3129 teilung1 = 300
3130 teilung2 = 300
3131
3132 deltaPsi = np.arange(-maxAnglePsi, maxAnglePsi, 2*maxAnglePsi/teilung1)
3133 deltaTheta = np.arange(-maxAngleTheta, maxAngleTheta,
     ↪  2*maxAngleTheta/teilung2)
3134
3135 R = 5
3136 roiX = pos2[100000:200000].x.max()-R
3137 roiY = pos2.iloc[pos2[100000:200000].x.argmax()].y-R
3138 roiZ = pos2[100000:200000].z.min() + R
3139 ROI = pos2.loc[ (pos2.x-roiX)**2+(pos2.y-roiY)**2 + (pos2.z-roiZ)**2< R**2]
3140 ROI_deltas = getDeltasSDMLarge(ROI.loc[:,['x','y','z']].get_values(),100)
3141
3142 #raster, psi, theta = searchMax(getDeltasSDMLarge(ROI.loc[:,['x','y','z']
     ↪  ].get_values(),100),deltaPsi,deltaTheta)
3143 print('psi theta ', psi, " ", theta)
```

```
3144 X,Y = np.meshgrid(deltaPsi,deltaTheta)
3145
3146
3147 aufl = 0.005
3148 binSize = 2.*deltaZMax/aufl
3149
3150 deltasAl = getDeltasSDMLarge(pos2.loc[pos2.comp ==
     ↪  'Al:1',['x','y','z']].get_values(), 500)
3151 zSDM_Al, w_zSDM_Al = getRotateZSDM(deltaZMax,deltasAl,psi,theta,binSize)
3152 zSDM_Mg, w_zSDM_Mg = getRotateZSDM(deltaZMax,deltasMg,psi,theta,binSize)
3153 zSDM_Si, w_zSDM_Si = getRotateZSDM(deltaZMax,deltasSi,psi,theta,binSize)
3154 zSDM_Cu, w_zSDM_Cu = getRotateZSDM(deltaZMax,deltasCu,psi,theta,binSize)
3155
3156 zSDM_ROI, w_zSDM_ROI = getRotateZSDM(deltaZMax,ROI_deltas,psi,theta,binSize)
3157
3158 [ax.clear() for ax in axes]
3159 CS = axes[5].contourf(X.T,Y.T, raster)
3160 axes[5].plot(psi,theta,'x')
3161 N0 = 70
3162 N0 = 200
3163
3164 axes[0].plot(zSDM_Al,normsmooth(w_zSDM_Al,N0), label='Al')
3165 N1 = 5
3166 axes[0].plot(zSDM_Al,normsmooth(w_zSDM_Mg,N1), label='Mg')
3167 axes[0].plot(zSDM_Al,normsmooth(w_zSDM_Si,N1), label='Si')
3168 #axes[0].plot(zSDM_Al,normsmooth(w_zSDM_Cu,N1), label='Cu')
3169
3170 axes[1].plot(zSDM_Al,norm(w_zSDM_Al))
3171 axes[1].plot(zSDM_Al,norm(w_zSDM_Mg))
3172 axes[1].plot(zSDM_Al,norm(w_zSDM_Si))
3173 #axes[1].plot(zSDM_Al,norm(w_zSDM_Cu))
3174
3175 axes[3].plot(zSDM_Al,normsmooth(w_zSDM_ROI,N0), label='Al')
3176
3177 deltasMgCluster = getDeltasSDMLarge(cluster.loc[cluster.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),2500)
3178 deltasSiCluster = getDeltasSDMLarge(cluster.loc[cluster.comp ==
     ↪  'Si:1',['x','y','z']].get_values(),2500)
3179 deltasAlCluster = getDeltasSDMLarge(cluster.loc[cluster.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3180
3181 zSDM_Mg_Cluster, w_zSDM_Mg_Cluster =
     ↪  getRotateZSDM(deltaZMax,deltasMgCluster,psi,theta,binSize)
3182 zSDM_Si_Cluster, w_zSDM_Si_Cluster =
     ↪  getRotateZSDM(deltaZMax,deltasSiCluster,psi,theta,binSize)
```

```
zSDM_Al_Cluster, w_zSDM_Al_Cluster =
↪ getRotateZSDM(deltaZMax,deltasAlCluster,psi,theta,binSize)

axes[4].clear()
N1 = 6
axes[4].plot(zSDM_Mg_Cluster,normsmooth(w_zSDM_Mg_Cluster,N1))
axes[4].plot(zSDM_Si_Cluster,normsmooth(w_zSDM_Si_Cluster,N1))

[ax.legend(loc='best') for ax in axes]
```

8.10 C14_art.py

```
from pyflann import *
from apt_importers import *
from Vis import volvis

vectors = np.loadtxt('/media/phillip/Volume1/Daten7050/C14_POSCAR_py2')

vectorA = vectors[0,:]/10.
vectorB = vectors[1,:]/10.
vectorC = vectors[2,:]/10.

#AB2_hP12_194_f_ah params=5.223,1.64005360904,0.06286,0.83048 SG#=194
↪ [ANRL doi: arXiv:1607.02532]
#1.000000
#   2.61150000000000  -4.52325068396612   0.00000000000000
#   2.61150000000000   4.52325068396612   0.00000000000000
#   0.00000000000000   0.00000000000000   8.56600000001592
#
#4 8
#Direct(12) [A4B8]
#   0.33333333333333   0.66666666666667   0.06286000000000
#   0.66666666666667   0.33333333333333   0.56286000000000
#   0.66666666666667   0.33333333333333  -0.06286000000000
#   0.33333333333333   0.66666666666667   0.43714000000000
#   0.00000000000000   0.00000000000000   0.00000000000000
#   0.00000000000000   0.00000000000000   0.50000000000000
#   0.83048000000000   0.66096000000000   0.25000000000000
#  0.33904  0.16952   0.25000000000000
#   0.83048000000000  0.16952   0.25000000000000
#  0.16952  0.33904   0.75000000000000
#   0.66096000000000   0.83048000000000   0.75000000000000
#  0.16952   0.83048000000000   0.75000000000000


point = vectors[3:,:]
```

```
3224
3225 origin = np.array([0., 0., 0.])
3226 n=10
3227
3228 C14 = getLattice(origin, vectorA, vectorB, vectorC, point, n, n, n)
3229 pos = pd.DataFrame({'x': C14[:,0],
3230                     'y': C14[:,1],
3231                     'z': C14[:,2],
3232                     'Da': 32.*np.ones(C14[:,0].size)})
3233 pos[0::12].Da = 24.
3234 pos[1::12].Da = 24.
3235 pos[2::12].Da = 24.
3236 pos[3::12].Da = 24.
3237
3238
3239 pathRange='/media/phillip/Volume1/13_rangeFiles/R21_09743_v01.rrng'
3240 ions,rrngs = read_rrng(pathRange)
3241 pos = label_ions(pos, rrngs)
3242 volvis(pos,size=7)
3243
3244 numberPlots = 4
3245 fig, axes = plt.subplots(1,numberPlots, figsize = (20,5))
3246 axes = axes.flatten()
3247
3248 psi = 0.
3249 theta = 0.
3250
3251 axes[0].clear()
3252 axes[1].clear()
3253 axes[2].clear()
3254 axes[3].clear()
3255
3256 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
    ↪  'Zn:1',['x','y','z']].get_values(),500)
3257 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3258 axes[0].plot(z_SDM_Al,norm(w_zSDM_Al),label='Zn')
3259 axes[0].legend(loc='best')
3260 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
    ↪  'Mg:1',['x','y','z']].get_values(),500)
3261 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3262 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mg',c='g')
3263 axes[1].legend(loc='best')
3264
3265 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
    ↪  'Zn:1',['x','y','z']].get_values(),500)
3266 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
```

```
3267 axes[2].plot(z_SDM_Al,norm(w_zSDM_Al),label='Zn',c='b')
3268
3269 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),500)
3270 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3271 axes[2].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mg',c='g')
3272 axes[2].legend(loc='best')
3273
3274 temp = read_epos('/media/phillip/Volume1/Daten7050/R21_09743/auswertung/pol
     ↪  e111_cluster_screen.epos')
3275 temp = label_ions(temp,rrngs)
3276
3277 psi = -0.27396263401595444
3278 theta = -0.34999999999999976
3279
3280 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3281 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3282 axes[0].plot(z_SDM_Al,10*norm(w_zSDM_Al),label='Zncluster_screen', c='r')
3283 axes[0].legend(loc='best')
3284
3285 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),300)
3286 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3287 axes[1].plot(z_SDM_Al,10*norm(w_zSDM_Al),label='Mgcluster_screen', c='r')
3288 axes[1].legend(loc='best')
3289
3290 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3291 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3292 axes[3].plot(z_SDM_Al, norm(w_zSDM_Al),label='Zncluster_screen')
3293 axes[3].legend(loc='best')
3294
3295 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),300)
3296 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3297 axes[3].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mgcluster_screen')
3298 axes[3].legend(loc='best')
3299
3300 [ax.set_xlabel(r'$\Delta$ z [nm]')for ax in axes]
3301 [ax.set_ylabel(r'fraction')for ax in axes]
3302 fig
3303
3304 numberPlots = 2
3305 fig, axes = plt.subplots(1,numberPlots, figsize = (10,5))
3306 axes = axes.flatten()
```

```
#[ax.set_xlabel(r'£\Delta£ x [nm]')for ax in axes]
#[ax.set_ylabel(r'£\Delta£ y [nm]')for ax in axes]

psi = 0.
theta = 0.
axes[2].clear()
deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
↪  'Zn:1',['x','y','z']].get_values(),500)
stutz2D, H2D = getRotateXYSDM(2, 2, deltas_Al, psi, theta, deltaZ=0.,
↪  deltaDeltaZ=0.01, binSDM=100)
X,Y = np.meshgrid(*stutz2D)
axes[2].contourf(X,Y,H2D,50)
axes[2].set_title('XY Zn')

axes[3].clear()
deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
↪  'Mg:1',['x','y','z']].get_values(),500)
stutz2D, H2D = getRotateXYSDM(2, 2, deltas_Al, psi, theta, deltaZ=0.,
↪  deltaDeltaZ=0.01, binSDM=100)
X,Y = np.meshgrid(*stutz2D)
axes[3].contourf(X,Y,H2D,50)
axes[3].set_title('XY Mg')
```

8.11 script_SDM_auswertung.py

```
maxAnglePsi = np.pi/180.*8
maxAngleTheta = np.pi/180.*8
teilung1 = 20
teilung2 = 20


deltaPsi = np.arange(-maxAnglePsi, maxAnglePsi, 2*maxAnglePsi/teilung1)
deltaTheta = np.arange(-maxAngleTheta, maxAngleTheta,
↪  2*maxAngleTheta/teilung2)
delta_temp =
↪  getDeltasSDMLarge(pole002.loc[:,['x','y','z']].get_values(),100)

raster, psi, theta = searchMax(delta_temp,deltaPsi,deltaTheta)
X,Y = np.meshgrid(deltaPsi,deltaTheta)
axes[0].contourf(X.T,Y.T, raster)

deltaPsi = np.arange(-maxAnglePsi+psi, maxAnglePsi+psi,
↪  2*maxAnglePsi/teilung1)
deltaTheta = np.arange(-maxAngleTheta+theta, maxAngleTheta+theta,
↪  2*maxAngleTheta/teilung2)
raster, psi, theta = searchMax(delta_temp,deltaPsi,deltaTheta)
```

```
3342 X,Y = np.meshgrid(deltaPsi,deltaTheta)
3343 axes[0].contourf(X.T,Y.T, raster)
3344
3345 deltas_Al = getDeltasSDMLarge(pole002.loc[pole002.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3346 # 100 -> NN
3347 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3348 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3349
3350 deltas_Zn = getDeltasSDMLarge(pole002.loc[pole002.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3351 z_SDM_Zn, w_zSDM_Zn = getRotateZSDM(1.,deltas_Zn,psi,theta,4./0.005)
3352 axes[1].plot(z_SDM_Al,norm(w_zSDM_Zn),label='Zn')
3353
3354 deltas_Mg = getDeltasSDMLarge(pole002.loc[pole002.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3355 z_SDM_Mg, w_zSDM_Mg = getRotateZSDM(1.,deltas_Mg,psi,theta,4./0.005)
3356 axes[1].plot(z_SDM_Mg,norm(w_zSDM_Mg),label='Mg')
3357
3358 deltas_Cu = getDeltasSDMLarge(pole002.loc[pole002.comp ==
     ↪  'Cu:1',['x','y','z']].get_values(),100)
3359 z_SDM_Cu, w_zSDM_Cu = getRotateZSDM(1.,deltas_Cu,psi,theta,4./0.005)
3360 axes[1].plot(z_SDM_Cu,norm(w_zSDM_Cu),label='Cu')
3361 #-----------------------------------------------------------------------
3362
3363 deltas_Al = getDeltasSDMLarge(cluster_pole002.loc[cluster_pole002.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3364 # 100 -> NN
3365 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3366 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3367
3368 deltas_Zn = getDeltasSDMLarge(cluster_pole002.loc[cluster_pole002.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3369 z_SDM_Zn, w_zSDM_Zn = getRotateZSDM(1.,deltas_Zn,psi,theta,4./0.005)
3370 axes[1].plot(z_SDM_Al,norm(w_zSDM_Zn),label='Zn')
3371
3372 deltas_Mg = getDeltasSDMLarge(cluster_pole002.loc[cluster_pole002.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3373 z_SDM_Mg, w_zSDM_Mg = getRotateZSDM(1.,deltas_Mg,psi,theta,4./0.005)
3374 axes[1].plot(z_SDM_Mg,norm(w_zSDM_Mg),label='Mg')
3375
3376 #-----------------------------------------------------------------------
3377 maxAnglePsi = np.pi/180.*8
3378 maxAngleTheta = np.pi/180.*8
3379 teilung1 = 20
3380 teilung2 = 20
```

```
3381
3382
3383 deltaPsi = np.arange(-maxAnglePsi, maxAnglePsi, 2*maxAnglePsi/teilung1)
3384 deltaTheta = np.arange(-maxAngleTheta, maxAngleTheta,
     ↪  2*maxAngleTheta/teilung2)
3385 delta_temp =
     ↪  getDeltasSDMLarge(pole111.loc[:,['x','y','z']].get_values(),100)
3386 raster, psi, theta = searchMax(delta_temp,deltaPsi,deltaTheta)
3387 X,Y = np.meshgrid(deltaPsi,deltaTheta)
3388 axes[0].contourf(X.T,Y.T, raster)
3389
3390 deltaPsi = np.arange(-maxAnglePsi+psi, maxAnglePsi+psi,
     ↪  2*maxAnglePsi/teilung1)
3391 deltaTheta = np.arange(-maxAngleTheta+theta, maxAngleTheta+theta,
     ↪  2*maxAngleTheta/teilung2)
3392 raster, psi, theta = searchMax(delta_temp,deltaPsi,deltaTheta)
3393 X,Y = np.meshgrid(deltaPsi,deltaTheta)
3394 axes[0].contourf(X.T,Y.T, raster)
3395
3396 deltas_Al = getDeltasSDMLarge(pole111.loc[pole111.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3397 # 100 -> NN
3398 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3399 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3400
3401 deltas_Zn = getDeltasSDMLarge(pole111.loc[pole111.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3402 z_SDM_Zn, w_zSDM_Zn = getRotateZSDM(1.,deltas_Zn,psi,theta,4./0.005)
3403 axes[1].plot(z_SDM_Al,norm(w_zSDM_Zn),label='Zn')
3404
3405 deltas_Mg = getDeltasSDMLarge(pole111.loc[pole111.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3406 z_SDM_Mg, w_zSDM_Mg = getRotateZSDM(1.,deltas_Mg,psi,theta,4./0.005)
3407 axes[1].plot(z_SDM_Mg,norm(w_zSDM_Mg),label='Mg')
3408
3409 #-------------------------------------------------------------------------
3410 axes[1].clear()
3411 cluster_pole111 = read_epos('/media/phillip/Volume1/Daten7050/R21_09743/aus
     ↪  wertung/cluster_pole111.epos')
3412 cluster_pole111 = label_ions(cluster_pole111,rrngs)
3413
3414 deltas_Al = getDeltasSDMLarge(cluster_pole111.loc[cluster_pole111.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3415 # 100 -> NN
3416 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3417 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
```

```
3418
3419 deltas_Zn = getDeltasSDMLarge(cluster_pole111.loc[cluster_pole111.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3420 z_SDM_Zn, w_zSDM_Zn = getRotateZSDM(1.,deltas_Zn,psi,theta,4./0.005)
3421 axes[1].plot(z_SDM_Al,norm(w_zSDM_Zn),label='Zn')
3422
3423 deltas_Mg = getDeltasSDMLarge(cluster_pole111.loc[cluster_pole111.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3424 z_SDM_Mg, w_zSDM_Mg = getRotateZSDM(1.,deltas_Mg,psi,theta,4./0.005)
3425 axes[1].plot(z_SDM_Mg,norm(w_zSDM_Mg),label='Mg')
3426
3427 temp = cluster_pole111[(cluster_pole111.z<60)&(cluster_pole111.z>50)]
3428 temp = temp[temp.y>19]
3429
3430 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3431 # 100 -> NN
3432 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3433 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3434
3435 deltas_Zn = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3436 z_SDM_Zn, w_zSDM_Zn = getRotateZSDM(2.,deltas_Zn,psi,theta,4./0.005)
3437 axes[1].plot(z_SDM_Al,norm(w_zSDM_Zn),label='Zn')
3438
3439 deltas_Mg = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3440 z_SDM_Mg, w_zSDM_Mg = getRotateZSDM(2.,deltas_Mg,psi,theta,4./0.005)
3441 axes[1].plot(z_SDM_Mg,norm(w_zSDM_Mg),label='Mg')
3442
3443 axes[1].axvline(z_SDM_Al[z_SDM_Al>0.1][w_zSDM_Al[z_SDM_Al>0.1].argmax()],c=
     ↪  'r',label='0.34')
3444 axes[1].legend()
3445
3446 #-----------------------------------------------------------------------
3447
3448 axes[1].clear()
3449 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3450 # 100 -> NN
3451 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3452 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Alcluster_screen')
3453
3454 deltas_Al = getDeltasSDMLarge(pole111.loc[pole111.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3455 # 100 -> NN
```

```
3456 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3457 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Alpole')
3458 axes[1].legend()
3459
3460 #-------------------------------------------------------------------------
3461
3462 axes[1].clear()
3463 pole111 = read_epos('/media/phillip/Volume1/Daten7050/R21_09743/auswertung/
     ↪ pole111.epos')
3464 pole111 = label_ions(pole111,rrngs)
3465
3466 maxAnglePsi = np.pi/180.*8
3467 maxAngleTheta = np.pi/180.*8
3468 teilung1 = 20
3469 teilung2 = 20
3470
3471 psi = -0.26
3472 theta = -0.35
3473 delta_temp =
     ↪ getDeltasSDMLarge(pole111.loc[:,['x','y','z']].get_values(),100)
3474
3475 deltaPsi = np.arange(-maxAnglePsi+psi, maxAnglePsi+psi,
     ↪ 2*maxAnglePsi/teilung1)
3476 deltaTheta = np.arange(-maxAngleTheta+theta, maxAngleTheta+theta,
     ↪ 2*maxAngleTheta/teilung2)
3477 raster, psi, theta = searchMax(delta_temp,deltaPsi,deltaTheta)
3478 X,Y = np.meshgrid(deltaPsi,deltaTheta)
3479 axes[0].contourf(X.T,Y.T, raster)
3480
3481 # pole 111
3482 # psi = -0.2739626340159544
3483 # theta = -0.34999999999999976
3484
3485 deltas_Al = getDeltasSDMLarge(pole111_matrix.loc[pole111_matrix.comp ==
     ↪ 'Al:1',['x','y','z']].get_values(),100)
3486 # 100 -> NN
3487 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3488 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3489
3490 axes[1].axvline(z_SDM_Al[z_SDM_Al>0.1][w_zSDM_Al[z_SDM_Al>0.1].argmax()],c=
     ↪ 'r',label='0.3')
3491
3492 temp = read_epos('/media/phillip/Volume1/Daten7050/R21_09743/auswertung/pol
     ↪ e111_cluster_screen.epos')
3493 temp = label_ions(temp,rrngs)
3494
```

```
3495 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3496 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3497 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Alcluster_screen')
3498
3499 #-----------------------------------------------------------------------
3500
3501 deltas_Al = getDeltasSDMLarge(pole002_cl.loc[pole002_cl.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3502 # 100 -> NN
3503 z_SDM_Al, w_zSDM_Al = getRotateZSDM(1.,deltas_Al,psi,theta,4./0.005)
3504 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3505
3506 deltas_Zn = getDeltasSDMLarge(pole002_cl.loc[pole002_cl.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3507 z_SDM_Zn, w_zSDM_Zn = getRotateZSDM(1.,deltas_Zn,psi,theta,4./0.005)
3508 axes[1].plot(z_SDM_Al,norm(w_zSDM_Zn),label='Zn')
3509
3510 deltas_Mg = getDeltasSDMLarge(pole002_cl.loc[pole002_cl.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3511 z_SDM_Mg, w_zSDM_Mg = getRotateZSDM(1.,deltas_Mg,psi,theta,4./0.005)
3512 axes[1].plot(z_SDM_Mg,norm(w_zSDM_Mg),label='Mg')
3513
3514 axes[1].legend(loc='best')
3515
3516 #-----------------------------------------------------------------------
3517
3518 axes[1].clear()
3519 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),100)
3520 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3521 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3522
3523 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),100)
3524 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3525 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Zn')
3526
3527 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),100)
3528 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3529 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mg')
3530 axes[1].set_title('cluster_screen')
3531 axes[1].legend(loc='best')
3532
3533 #-----------------------------------------------------------------------
```

```
3534
3535 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),500)
3536 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3537 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Al')
3538
3539 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3540 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3541 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Zn')
3542
3543 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),300)
3544 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3545 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mg')
3546
3547 axes[0].clear()
3548 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Al:1',['x','y','z']].get_values(),500)
3549 stutz2D, H2D = getRotateXYSDM(1, 1, deltas_Al, psi, theta, deltaZ=0.,
     ↪  deltaDeltaZ=0.1, binSDM=100)
3550 X,Y = np.meshgrid(*stutz2D)
3551 axes[0].contourf(X,Y,H2D,50)
3552 fig
3553 #--------------------------------------------------------------------
3554
3555 psi = 0.
3556 theta = 0.
3557
3558 axes[0].clear()
3559 axes[1].clear()
3560 axes[2].clear()
3561 axes[3].clear()
3562
3563 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3564 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3565 axes[0].plot(z_SDM_Al,norm(w_zSDM_Al),label='Zn')
3566 axes[0].legend(loc='best')
3567 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),500)
3568 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3569 axes[1].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mg',c='g')
3570 axes[1].legend(loc='best')
3571
```

```
3572 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3573 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3574 axes[2].plot(z_SDM_Al,norm(w_zSDM_Al),label='Zn',c='b')
3575
3576 deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),500)
3577 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3578 axes[2].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mg',c='g')
3579 axes[2].legend(loc='best')
3580
3581 temp = read_epos('/media/phillip/Volume1/Daten7050/R21_09743/auswertung/pol
     ↪  e111_cluster_screen.epos')
3582 temp = label_ions(temp,rrngs)
3583
3584 psi = -0.27396263401595444
3585 theta = -0.34999999999999976
3586
3587 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3588 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3589 axes[0].plot(z_SDM_Al,10*norm(w_zSDM_Al),label='Zncluster_screen', c='r')
3590 axes[0].legend(loc='best')
3591
3592 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),300)
3593 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3594 axes[1].plot(z_SDM_Al,10*norm(w_zSDM_Al),label='Mgcluster_screen', c='r')
3595 axes[1].legend(loc='best')
3596
3597 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Zn:1',['x','y','z']].get_values(),500)
3598 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3599 axes[3].plot(z_SDM_Al, norm(w_zSDM_Al),label='Zncluster_screen')
3600 axes[3].legend(loc='best')
3601
3602 deltas_Al = getDeltasSDMLarge(temp.loc[temp.comp ==
     ↪  'Mg:1',['x','y','z']].get_values(),300)
3603 z_SDM_Al, w_zSDM_Al = getRotateZSDM(2.,deltas_Al,psi,theta,4./0.005)
3604 axes[3].plot(z_SDM_Al,norm(w_zSDM_Al),label='Mgcluster_screen')
3605 axes[3].legend(loc='best')
3606
3607 [ax.set_xlabel(r'$\Delta$ z [nm]')for ax in axes]
3608 [ax.set_ylabel(r'fraction')for ax in axes]
3609 fig
3610
```

```
psi = 0.
theta = 0.
axes[2].clear()
deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
↪  'Zn:1',['x','y','z']].get_values(),500)
stutz2D, H2D = getRotateXYSDM(2, 2, deltas_Al, psi, theta, deltaZ=0.,
↪  deltaDeltaZ=0.01, binSDM=100)
X,Y = np.meshgrid(*stutz2D)
axes[2].contourf(X,Y,H2D,50)
axes[2].set_title('XY Zn')

axes[3].clear()
deltas_Al = getDeltasSDMLarge(pos.loc[pos.comp ==
↪  'Mg:1',['x','y','z']].get_values(),500)
stutz2D, H2D = getRotateXYSDM(2, 2, deltas_Al, psi, theta, deltaZ=0.,
↪  deltaDeltaZ=0.01, binSDM=100)
X,Y = np.meshgrid(*stutz2D)
axes[3].contourf(X,Y,H2D,50)
axes[3].set_title('XY Mg')
```

8.12 clusteranalyse_AlMgSi.m

```
pos = readpos;
rng = {};
%copy Excel rng data

readXLSRanges(rng,pos,'ionic');
[pass, Nmin, clusterCutoff, clusteredAtoms] =
↪  clusterDetermination([Mgpos;Sipos],pos);
clusterNumbers = unique(clusteredAtoms(:,5));
%delete ranged Atoms


figure()
hold on
shapes = cell([length(clusterNumbers),1]);
for i = 1:length(clusterNumbers)
    shapes{i} = alphaShape(clusteredAtoms(clusteredAtoms(:,5) ==
    ↪  clusterNumbers(i),1:3));
    plot(alphaShape(clusteredAtoms(clusteredAtoms(:,5) ==
    ↪  clusterNumbers(i),1:3)));
end

atomsFound = [];
for i = 1:length(clusterNumbers)
    positionen = pos(inShape(shapes{i},pos(:,1), pos(:,2), pos(:,3)),:);
```

```
3647     atomsFound = [atomsFound;[positionen,i*ones([length(positionen),1])]];
3648 end
3649
3650 scatter3(atomsFound(:,1),atomsFound(:,2),atomsFound(:,3),'.');axis equal;
3651
3652 %  readXLSRanges(rng,atomsFound,'ionic');
3653
3654 %  ----------------- Al H Mg Si Cu Ga
3655 %  . 13.4500 13.6000 1 0 0 0 0 0
3656 %  . 26.9210 27.0990 1 0 0 0 0 0
3657 %  . 0.9850 1.0950 0 1 0 0 0 0
3658 %  . 1.9790 2.1090 0 2 0 0 0 0
3659 %  . 11.9550 12.0500 0 0 1 0 0 0
3660 %  . 12.4560 12.5750 0 0 1 0 0 0
3661 %  . 12.9430 13.1070 0 0 1 0 0 0
3662 %  . 13.9530 14.0790 0 0 0 1 0 0
3663 %  . 14.4490 14.5490 0 0 0 1 0 0
3664 %  . 14.9530 15.0440 0 0 0 1 0 0
3665 %  . 62.8120 63.1100 0 0 0 0 1 0
3666 %  . 64.8100 65.1500 0 0 0 0 1 0
3667 %  . 27.9400 28.1760 1 1 0 0 0 0
3668 %  . 28.9280 29.2410 1 2 0 0 0 0
3669 %  . 23.2550 23.4340 0 1 0 0 0 1
3670
3671 max = 0;
3672 idC = 0;
3673 for i = 1 : length(unique(atomsFound(:,5)))
3674         if max < length(atomsFound(atomsFound(:,5) == i))
3675                 max = length(atomsFound(atomsFound(:,5) == i));
3676                 idC = i;
3677         end
3678 end
3679
3680 readXLSRanges(rng,atomsFound(atomsFound(:,5)~=idC,1:5),'ionic');
3681
3682 %  summe =
     ↪  length([Alpos;AlH2pos;AlHpos;Cupos;H2pos;Hpos;Mgpos;Sipos,HGapos]);
3683 summe = length([Alpos;AlH2pos;AlHpos;Cupos;Mgpos;Sipos]);
3684 [(length(Alpos)+length(AlH2pos)+length(AlHpos))/summe, length(Cupos)/summe,
     ↪  length(Mgpos)/summe, length(Sipos)/summe]
```

8.13 motorsteuerung_v09.ino

```
3685 // init constants
3686 #include <AccelStepper.h>
```

```
3687 const int onOffSwitchStateSwitchPin = 5; // connected to the switch for
     ↪  turning the motor on and off
3688 const int raufPin = 11; // connected to the switch for moving upwards
3689 const int runterPin = 10; // connected to the switch for moving downwards
3690 const int sekStromPin = 7; // switch for electrolysis circuit
3691 const int inVoltage = A0; // analog measurement of voltage drop at shunt
     ↪  resistor
3692 const int LED_pin = 4; // pin zur LED verbunden
3693 const char HEADER = 'H';
3694
3695 const int FILTER_SHIFT = 4;
3696
3697 // init variables
3698
3699 int onOffSwitchState = 0;  // current state of the On/Off switch
3700 int previousOnOffSwitchState = 0; // previous position of the on/off
     ↪  switch
3701
3702 int raufSwitch = 0;
3703 int runterSwitch = 0;
3704
3705 int current = 0.;
3706
3707 float mean = 0;
3708 float diff = 0.;
3709 float voltageDiv = 5.7;
3710 float R = 2.2;
3711 float convmA = 1000.;
3712 float maxcurrent = 0.;
3713 const float currentlim = 2.3;
3714 const float maxVoltage = 5.;
3715 float data[4];
3716
3717 unsigned long timepreviousData = 0;
3718 unsigned long timecurrentData = 0;
3719
3720 long filter_reg = 0;
3721 int filter_out = 0;
3722 unsigned long timepreviousDataAnalog = 0;
3723
3724
3725 int abtastintervall = 25;
3726 int abtastintervallAnalog = 3;
3727 int values;
3728
3729 int counter = 0;
```

```
3730 int countVal = 0;
3731
3732 //-----------------------------------------------------------------------
3733 unsigned long t1;
3734 unsigned long t;
3735 unsigned long dt;
3736 unsigned long maxdt;
3737
3738 float i;
3739
3740 float iUp;
3741 float iDown;
3742 float iUp0;
3743 float iDown0;
3744 float iHard;
3745
3746 float f;
3747 float k;
3748 float const cKrit = 0.5;
3749 float konstante = -1.;
3750 float iCoeff = 0.;
3751
3752 float maximumSpeed = 1000.;
3753 float maxAcc = 200.;
3754 float speed0 = 266.;
3755
3756 long positionSteps = 15;
3757
3758 unsigned long start;
3759
3760 AccelStepper stepper(AccelStepper::FULL4WIRE, 8, 9,3,2,true);
3761
3762 boolean initial = true;
3763 boolean richtung = true;
3764
3765 boolean iDownSet = false;
3766 boolean iUpSet = false;
3767 boolean state0 = false;
3768 boolean sendenBool = true;
3769 boolean stromAktiv = false;
3770 //-----------------------------------------------------------------------
3771 void setup() {
3772   // intialize the inputs and outputs
3773 //  Serial.begin(115200);
3774   Serial.begin(115200);
3775   pinMode(onOffSwitchStateSwitchPin, INPUT);
```

```
3776   pinMode(raufPin, INPUT);
3777   pinMode(runterPin, INPUT);
3778   pinMode(sekStromPin, OUTPUT);
3779   pinMode(LED_pin,OUTPUT);
3780 
3781 // pull the enable pin LOW to start
3782 //  digitalWrite(enablePin, LOW);
3783 //  digitalWrite(sekStromPin,LOW);
3784   digitalWrite(sekStromPin,LOW);
3785   digitalWrite(LED_pin,LOW);
3786 
3787   dt = 0;
3788   maxdt = 0.;
3789   iHard = 0.;
3790 
3791 //  t1 = 2*sqrt(2.*positionSteps/maxAcc);
3792 //  t1 = 1900;
3793   t1 = 2*positionSteps;
3794 
3795   iCoeff = 5./1024.*convmA/R;
3796 
3797   iDown = 10.;
3798   iUp = 5.;
3799   f = 3.;
3800   i = 7.;
3801 
3802   stepper.setMaxSpeed(maximumSpeed); // steps per second
3803   stepper.setAcceleration(maxAcc); // steps per second squared
3804   stepper.moveTo(positionSteps);
3805 
3806   richtung = true;
3807 
3808 }
3809 //--------------------------------------------------------------------
3810 void loop() {
3811   // read the value of the on/off switch
3812   onOffSwitchState = digitalRead(onOffSwitchStateSwitchPin);
3813 
3814   timecurrentData = millis();
3815   if ((unsigned long)(timecurrentData - timepreviousDataAnalog) >=
      ↪  abtastintervallAnalog){
3816   filter_reg = filter_reg - (filter_reg >> FILTER_SHIFT) +
      ↪  analogRead(inVoltage);
3817   filter_out = filter_reg >> FILTER_SHIFT;
3818   timepreviousDataAnalog = timecurrentData;
3819   }
```

```
3820 //-------------------------------------------------------------------------D
     ↪ EBUGGING
3821   if (((unsigned long)(timecurrentData - timepreviousData) >=
       ↪ abtastintervall) && sendenBool){
3822   Serial.write(HEADER);
3823
3824   values = (int)(timecurrentData-timepreviousData);
3825   sendBinary(values);
3826   sendBinary(filter_out);
3827   timepreviousData = timecurrentData;
3828   counter+=1;
3829   if (counter > 27){
3830     counter = 0;
3831     if (filter_out < countVal){
3832       if (countVal - filter_out > countVal/2){
3833         // stopp
3834 //         sendenBool = false;
3835       }
3836
3837     }
3838     countVal = filter_out;
3839   }
3840   }
3841 //   delay(1);
3842 //delayMicroseconds(100);
3843 //-------------------------------------------------------------------------D
     ↪ EBUGGING
3844
3845   // if the on/off button changed state since the last loop()
3846   if (onOffSwitchState != previousOnOffSwitchState) {
3847     // change the value of motorEnabled if pressed
3848     if (onOffSwitchState == HIGH) {
3849 //-------------------------------------------------------------------------s
     ↪ tart old
     ↪ controll
3850 /**state0 = !state0;
3851
3852       stepper.setCurrentPosition(0);
3853       stepper.setMaxSpeed(speed0); // steps per second
3854       stepper.setAcceleration(maxAcc); // steps per second squared
3855       stepper.moveTo(positionSteps);
3856
3857       start = timecurrentData;
3858 /**
3859       initial = true;
3860       richtung = true;
```

```
3861        iDownSet = false;
3862        iUpSet = false;
3863  **/
3864  //-------------------------------------------------------------------e
      ↪  nd old
      ↪  controll
3865          stromAktiv = !stromAktiv;
3866          if(stromAktiv){
3867                  digitalWrite(sekStromPin,HIGH);
3868          }
3869          else{
3870                  digitalWrite(sekStromPin,LOW);
3871          }
3872      }
3873    }
3874
3875    if (state0){
3876    stage3();
3877    stepper.run();
3878    if (stepper.distanceToGo() == 0)
3879      {
3880        stepper.moveTo(-stepper.currentPosition());
3881        start = timecurrentData;
3882        changeCurrentBounds();
3883        richtung = !richtung;
3884      }
3885    }
3886    else{
3887  //  digitalWrite(sekStromPin,LOW);
3888  //  digitalWrite(sekStromPin,HIGH);
3889
3890    raufSwitch = digitalRead(raufPin);
3891    runterSwitch = digitalRead(runterPin);
3892      if( raufSwitch == HIGH ){
3893        stepper.setSpeed(speed0/4.);
3894       while(!stepper.runSpeed()){
3895       }
3896        }
3897      else{
3898        if( runterSwitch == HIGH ){
3899       stepper.setSpeed((-1)*speed0/4.);
3900       while(!stepper.runSpeed()){
3901       }
3902        }
3903      }
3904    }
```

```
3905   // save the current switch state as the previous
3906   previousOnOffSwitchState = onOffSwitchState;
3907 }
3908 void stage3(){
3909         if(initial)
3910         {
3911         // einschalten des Elektrolysestromes
3912                 digitalWrite(sekStromPin,HIGH);
3913                 initial = false;
3914         }
3915 //         t = timecurrentData;
3916 //         dt = t - start;
3917 //         if(dt>maxdt){
3918 //                 maxdt = dt;
3919 //         }
3920 //          previousCurrent = i;
3921         i = readCurrent();
3922 //         // UEBERWACHUNG Kriterium Elektropolieren
3923         f = fAbbr(i);
3924 //         if ((i<cKrit*fAbbr(i)) || (i < iHard)) {
3925 //                 state0 = false;
3926 //                 stepper.stop();
3927 //         // ENDE Elektropolieren
3928 //         }
3929 }
3930 float readCurrent(){
3931         current = analogRead(inVoltage);
3932         return current;
3933 }
3934 float fAbbr(float i){
3935         f = i;
3936
3937         if(iDownSet && iUpSet){
3938     t = stepper.currentPosition();
3939                 if (richtung){
3940       t = t+positionSteps;
3941                         if (iUp < iUp0){
3942                         k = iUp/iUp0;
3943                         f = (iUp*k-iDown)/ t1*t+ iDown;
3944                         }
3945                         else{
3946                                 f = (iUp-iDown)/t1*t+ iDown;
3947                         }
3948                 }
3949                 else{
3950    t = positionSteps-t;
```

```
3951                            if (iDown < iDown0){
3952                            k = iDown/iDown0;
3953                            f = iUp - (iUp-iDown*k)/t1*t;
3954                            }
3955                            else{
3956                                    f = iUp -(iUp-iDown)/t1*t;
3957                            }
3958                    }
3959            }
3960            return f;
3961    }
3962    void changeCurrentBounds(){
3963            if (stepper.currentPosition() < 0)
3964            {
3965                    iDown0 = iDown;
3966                    iDown = i;
3967                    iDownSet = true;
3968            }
3969            else
3970            {
3971                    iUp0 = iUp;
3972                    iUp = i;
3973                    iUpSet = true;
3974            }
3975    }
3976    void sendBinary( int value)
3977    {
3978      // send the two bytes that comprise an integer
3979      Serial.write(lowByte(value));  // send the low byte
3980      Serial.write(highByte(value)); // send the high byte
3981    }
```